FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Smart Objects for Manufacturing

Daniel Filipe Figueiredo Costa

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Professor Américo Lopes de Azevedo (PhD)

July 24, 2014

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES 2013/2014

A Dissertação intitulada

"Smart Objects for Manufacturing"

foi aprovada em provas realizadas em 17-07-2014

o júri

Presidente Professor Doutor Fernando Arménio da Costa Castro e Fontes
Professor Associado do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Samuel de Oliveira Moniz
Assistente Convidado da Escola Superior de Ciências Empresariais do Instituto Politécnico de Viana do Castelo

Professor Doutor Américo Lopes de Azevedo
Professor Associado do Departamento de Engenharia de Gestão Industrial da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - Daniel Filipe Figueiredo Costa

Faculdade de Engenharia da Universidade do Porto

Resumo

Internet of Things (IoT). Este é o novo paradigma que promete mudar tudo como conhecemos atualmente e que vai transformar as nossas vidas numa rede global. Se considerarmos a IoT a evolução da Internet, que permite recolher, analisar e distribuir dados que podem ser processados como informação, conhecimento e consecutivamente sabedoria, a IoT torna-se exponencialmente importante.

Está envolvida uma grande quantidade de tecnologia no conceito de IoT e isto representa um desafio para implementar uma *framework* funcional. Não existem plataformas que incorporem todas as funcionalidades dos RFID e sensores. Há, no entanto, a possiblilidade de usar um Smart Object (SO) *framework* como uma arquitetura que inclua as funcionalidades mais importantes que se prentende: Smart Objects são objetos não só capazes de fornecer a sua identificação e condição única, mas também de realizar comunicações *object-to-object*, redes *ad-hoc* e decisões centradas no objeto. Terá de haver uma infrastrutura de informação que permita ao utilizador interagir com o SO e que disponibiliza o acesso ao sensor e a rede de informação do SO.

Existem muitas áreas onde a IoT pode ser implementada, mas neste caso irá ser abordado o seu uso com SO em maquinaria na manufactura industrial. O trabalho apresentado neste documento foi desenvolvido no contexto de um projeto europeu - projeto ADVENTURE (ADaptive Virtual ENterprise ManufacTURing Environment) que visa, essencialmente, a criação de uma estrutura que forneça ferramentas para interligar fábricas, na criação de um determinado produto. Isto incui a criação de processos de fabrico, encontrar parceiros, bem com a monitorização em tempo real dos processos activos.

Este SO deverá ser capaz de interligar o meio físico e exterior ao resto da plataforma, de forma a tornar esta solução numa solução completa. Deverá possibilitar ao utilizador a criação de regras, as quais irão ser posteriormente interpretadas e resultarão naquilo que esse mesmo utilizador pretender.

Através da criação de uma plataforma fisíca de implementação, da comunicação realizada e de toda a programação do seu comportamento, esta solução de Smart Object é capaz de implementar as funcionalidades da Internet of Things em ambiente industrial tal como: a monitorização em tempo real de maquinaria, configuração de regras e eventos e gerar ações resultantes do comportamento pretendido. Este projecto serve de ponto de partida para uma completa inclusão do SO no projecto ADVENTURE e exemplifica um avanço importante em tornar a indústria da manufatura cada vez mais inovadora, global e competitiva.

Abstract

The Internet of Things (IoT), this new paradigm that will change everything as we now know, can transform our lives into a global network. If we consider IoT the evolution of the Internet, by allowing us to gather, analyse, and distribute data that can be processed as information, knowledge and, therefore, wisdom, the IoT becomes exceptionally important.

It is involved a great amount of several technologies in the IoT concept and that represents the challenge to implement an effective integration framework. There are no existing platforms that incorporate the all the functionalities of the RFID and sensors. There is, however, the possibility to use the Smart Object (SO) framework as an architecture that features the most important functionalities that we look for: Smart Objects are objects not only capable of providing their unique identification and condition, but also able to perform object-to-object communications, ad-hoc networking and object-centric complex decision making. There as to be an information infrastructure that which enables the user to interact with that SO, and provides access to both ID/sensor and the SO network information.

There are many areas where the IoT can be implemented, in this case we will focus on the use of SO in machinery of the manufacturing industry. The work presented in this document was develop in the context of an European project - the ADVENTURE (ADaptive Virtual ENterprise ManufacTURing Environment) that aims the creation of a framework that provides the tools to combine factories in a pluggable way to manufacture a particular product. This includes the creation of manufacturing processes, finding partners as well as real-time monitoring of the processes that are put into play.

This SO should be able to connect the physical and external environment to the rest of the platform, in order to make this a complete solution. Should enable the user to create rules which will later be interpreted and will result in what the same user wanted.

Through the creation of a physical deployment platform, communications all operational and the core behaviour programmed, this solution of Smart Object is able to implement Internet of Things functionalities in an industrial environment such as: real-time monitoring of machinery, rules setting and generate events and actions arising from the intended behaviour. This project can be a starting point for a complete inclusion of the SO in the ADVENTURE project and exemplifies an important advance in the manufacturing industry in becoming increasingly innovative, global and competitive.

Agradecimentos

Em primeiro lugar quero agradecer à minha família, em particular aos meus pais e irmã, por todo o apoio incondicional, amizade e motivação ao longo destes anos todos. Sem eles não estaria aqui, nem seria quem sou.

Depois agradecer a todos os que contribuíram diretamente e me ajudaram durante este projeto:

- Ao meu orientador Professor Doutor Américo Azevedo pela orientação do projeto. Pela disponibilidade e interesse demonstrados ao longo deste trabalho e por me direcionar quando não sabia qual o rumo a seguir;
- Aos Engs. António Almeida e Álvaro Caldas, por todo o apoio prestado durante a programação dos scripts eesclarecimento de dúvidas relacionadas com o projeto;
- Ao José Carlos Azevedo, pela ajuda na esquematização e produção da PCB.

Não esquecer também todos os professores que ao longo destes cinco anos me ensinaram e me ajudaram a atingir este grande objetivo na minha vida.

À Joaninha pelo carinho, boa disposição constante e prontidão em ajudar e motivar sempre que algo não corre como planeado.

Por último, mas não menos importante, quero agradecer a todos os meus amigos pela boa disposição, entre-ajuda e motivação, essenciais para a conclusão do curso e do projeto. Uma palavra especial ao "gang do I105" pelo incentivo extra e companheirismo exemplar nesta fase final da minha vida académica.

Daniel Filipe Figueiredo Costa

"Progress is impossible without change,
and those who cannot change their minds
cannot change anything."

George Bernard Shaw

Contents

Resumo iii

Abstract v

Agradecimentos vii

Abbreviations xix

1 Introduction 1
- 1.1 Presentation of the Problem . . . 1
- 1.2 Motivation . . . 2
- 1.3 Proposed Goals . . . 2
- 1.4 Methodology . . . 3
- 1.5 Structure . . . 3

2 State of the Art 5
- 2.1 ADVENTURE Project . . . 5
- 2.2 Internet of Things . . . 6
  - 2.2.1 Definition . . . 6
  - 2.2.2 Concept . . . 7
  - 2.2.3 Applications . . . 10
- 2.3 Smart Objects . . . 13
  - 2.3.1 Smart Solutions . . . 13
  - 2.3.2 Definition . . . 14
  - 2.3.3 Technology Background . . . 15
  - 2.3.4 Components . . . 16
  - 2.3.5 Existing Products . . . 25

3 Smart Object for Manufacturing 27
- 3.1 Concept . . . 27
- 3.2 Requirements . . . 28
- 3.3 Functionalities . . . 29
- 3.4 Rules . . . 30
  - 3.4.1 Discrete Rules . . . 30
  - 3.4.2 Continuous Rules . . . 31
  - 3.4.3 Internal Rules . . . 31
  - 3.4.4 Extra Rules . . . 32

4 Solution Design **35**
4.1 Concept . . . 35
4.1.1 Digital Inputs . . . 36
4.1.2 Analog Inputs . . . 36
4.1.3 Outputs . . . 36
4.2 Production . . . 36
4.2.1 Digital Inputs . . . 36
4.2.2 Analog Inputs . . . 37
4.2.3 Relay Controlled Outputs . . . 38
4.2.4 Real-time Clock . . . 40
4.2.5 Serial Port . . . 40
4.2.6 DC-DC Converter . . . 41
4.3 Printed Circuit Board . . . 41
4.4 Conclusion . . . 42

5 Communication **43**
5.1 BeagleBone Black - PC . . . 43
5.2 Switch Network . . . 43
5.2.1 BBB - PLC: FINS Protocol . . . 44
5.3 Conclusion . . . 45

6 Processing **47**
6.1 Introduction . . . 47
6.2 Python . . . 47
6.2.1 Libraries . . . 48
6.3 Database . . . 49
6.3.1 Rules . . . 49
6.3.2 Logs . . . 50
6.3.3 Timers . . . 51
6.4 Inputs . . . 52
6.5 Rules . . . 61
6.6 Conclusion . . . 67

7 Results **69**
7.1 Tests . . . 69
7.2 Conclusion . . . 74

8 Conclusions and Future Work **77**
8.1 Conclusions and Fulfilment of Goals . . . 77
8.2 Future Work . . . 79

A ADVENTURE **81**
A.1 User Interface . . . 81

B I/O Interface Schematics **83**
B.1 Printed Circuit Board . . . 83

C Code **87**
C.1 Omron Communication Code . . . 87
C.1.1 finscommunicator.cpp . . . 87
C.2 Processing Scripts . . . 91
C.2.1 inputs.py . . . 91
C.2.2 rules.py . . . 99

D User Visual Interface **105**
D.1 Inputs . . . 105
D.2 Rules . . . 106
D.2.1 Use Case . . . 107
D.3 Outputs . . . 108
D.4 Logs . . . 109

References **111**

List of Figures

2.1 The ADVENTURE Layer . . . 6
2.2 Internet of Things . . . 7
2.3 The IoT was "born" between 2008 and 2009 . . . 8
2.4 Gartner 2012 Hype Cycle of emerging technologies . . . 9
2.5 Google search trends since 2004 for terms Internet of Things, Wireless Sensor Networks, Ubiquitous Computing. . . . 9
2.6 Humans Turn Data into Wisdom . . . 10
2.7 Conceptual Layers of the IoT . . . 11
2.8 Applications of the IoT . . . 11
2.9 Areas with significant opportunities for Smart Solutions . . . 14
2.10 Participants in the value chain for Smart Solutions . . . 15
2.11 Smart Objects definition . . . 16
2.12 Smart Objects or "BlackBox" components . . . 16
2.13 Sensoring interactions . . . 17
2.14 OpenSprinkler . . . 19
2.15 Osso I/O . . . 20
2.16 MOD-IO . . . 21
2.17 BeagleBone Black . . . 22
2.18 Raspberry PI . . . 22
2.19 BeagleBone Black vs Raspberry PI . . . 23
2.20 Main communications envolving the SmartBox . . . 24
2.21 WRM® 365 . . . 25
2.22 WRM overview . . . 26

3.1 ADVENTURE components . . . 28

4.1 Concept Diagram . . . 35
4.2 Schematic diagram of an opto-isolator showing source of light (LED) on the left, dielectric barrier in the center, and sensor (phototransistor) on the right.[1] . . . 36
4.3 Digital Input . . . 37
4.4 Simulation of the Analogic Voltage Input . . . 38
4.5 Simulation of the Analogic Current Input . . . 38
4.6 Relay Output with ULN2003 . . . 39
4.7 Interface I/O-Beta version . . . 39
4.8 Real-time Clock . . . 40
4.9 TTL-RS232 Serial Port for P9 and J1 . . . 40
4.10 DC-DC Converter . . . 41
4.11 BeagleBoneBlack Top and Bottom Silk Screen . . . 42

4.12 Printed Circuit Board 42

5.1 Connection with BBB through SSH 43
5.2 Actual Switch Network 44

6.1 Example of Python script and result 48
6.2 Adafruit.GPIO Library example 49
6.3 Database Rules table 50
6.4 Database Logs table 51
6.5 Database Timers table 52

7.1 Result Condition 1 69
7.2 Result Condition 2 70
7.3 Result Condition 3 70
7.4 Result Condition 4 71
7.5 Result Condition 5 71
7.6 Result Condition 6 72
7.7 Result Condition 7 73
7.8 Result Condition 8 74

A.1 ADVENTURE Goal 82

D.1 Inputs Interface Screen 105
D.2 Rules Interface Screen 106
D.3 Possibilities on adding a new rule 106
D.4 Interface Use case - Adding a rule 107
D.5 Interface Use Case - Rule created 108
D.6 Outputs Interface Screen 108
D.7 Inputs Interface Screen 109

List of Tables

2.1 Summary of RFID and Sensor related work 17
2.2 Specifications of the OpenSprinkler - Beagle-Master 19

Abbreviations and Symbols

Abbreviations

ADC	Analog-Digital Converter
ADVENTURE	ADaptive Virtual ENterprise ManufacTURing Environment
BB	BlackBox
BBB	BeagleBone Black
EU	European Union
FINS	Factory Interface Network Service
FTP	File Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IoT	Internet of Things
M2M	Machine to Machine
MIT	Massachusetts Institute of Technology
NFC	Near Field Communication
OEE	Overall equipment effectiveness
PC	Personal Computer
PCB	Printed Circuit Board
PLC	Programmable Logic Controller
RFID	Radio-frequency identification
RPI	Raspberry PI
SO	Smart Object
SSH	Secure Shell
USB	Universal Serial Bus
WWW	World Wide Web

Symbols

I	Current
Vi	Input Voltage
Vo	Output Voltage
R	Resistor

Chapter 1

Introduction

1.1 Presentation of the Problem

The world economy is changing rapidly and unpredictably. This change, powered by the increasing globalisation of markets and emergence of developing economies on the one hand, and climate change, energy, food and health issues on the other hand, is challenging all businesses. Nowhere is this more keenly felt than amongst Europe's small manufacturing companies. In addition to the global challenges, that we all face, the specific challenges facing a typical EU manufacturing SME include:

- Large customers moving their emphasis and factories to the emerging markets of the East;
- Competitors from countries with extremely low costs of employment;
- Changing demographics of European customers and staff;
- The internet making customers better informed and more aware of what competitor are offering;
- Bureaucracy as a barrier to growth;
- Problems in finding skilled employees and managers;
- Fast technological change.

It is estimated that for the EU, 75% of Gross domestic product (GDP) and 70% of employment is related to manufacturing [1]. One out of four jobs in the private sector in the European Union is in the manufacturing industry, and at least another one out of four is in associated services that depend on industry as a supplier or as a client. There are about 2.5 million manufacturing SMEs in Europe that represent 99% of European manufacturing businesses. At the same time 80% of all private sector research and development efforts are undertaken in that industry. [2]

In his paper "Adaptive Capability - A must for manufacturing SMEs of the Future", Bititci et al (2012) explained: "It is widely recognised that manufacturing SMEs need to be able to quickly

and effectively reconfigure their production and other resources to keep up with the increasing pace of change. In fact, they need to do more than just change – they need to be able to shape the future, so innovation becomes very important.

Having all this in mind, the relation between the Internet and the manufacturing industry is inevitable. The Internet of Things (IoT), in which every object is connected to the Internet, is an obvious new direction and its application to the manufacturing field a revolution in all that we know. To be able to enjoy all its advantages there has to be an object that relates the Internet of Things with the machines - the Smart Object (SO). The purpose of this dissertation is to study, concept and implement a Smart Object framework, capable of satisfying all the goals presented bellow.

1.2 Motivation

It is a driver of innovation and a provider of solutions to the challenges our societies are confronted with. The future of manufacturing is part of the European economic growth and sustainability and EU competitiveness is strongly influenced by the performance of its manufacturing SMEs and large companies. There is an increasing demand for greener, more customised and higher quality products. The long-term shift, from a cost-based competitive advantage to one based on high added value, requires that European manufacturing increases its technological base and develops a number of new production technologies with benefits from all sectors [3]. This innovation and technological advances can only be implemented with the creation of newer solutions, for which the Internet and, therefore, the Internet of Things are a key resource. Although a functional and fully featured solution it is still missing, the concept of Smart Objects using the IoT is already in development and all the efforts to materialize it, like the one that motivates this project, are useful and needed.

1.3 Proposed Goals

The main goal of this dissertation is to conceive a solution to implement features from the Internet of Things at industrial equipment level. In order to accomplish that solution must have several components to connect to a complex framework and thus be able to implement IoT features. Exploring the IoT concept, at the industrial machinery level were found some of those features such as: acquire information from the solution exterior, access external information and process data at the operation and management levels. The division in phases was inevitable to help the organization and the flow throughout the project and that division can be considered as objectives inside the main goal:

- Explore the IoT concept, at the industrial machinery level;
- Detail the architecture and concept of the solution;
- Development and implementation of the prototype;

- Validation and conclusions.

1.4 Methodology

The development methodology applied in order to accomplish the main goal was composed by several steps. First it was conducted a bibliographic survey to introduce the concept of Internet of Things and Smart Objects and to ascertain if similar solutions already existed in the market. Right after that step, a concept in the application domain was developed, so that its implementation had all requirements needed. That implementation was then carried concerning three different parts and phases: sensoring, communication and processing. The first part related with the data gathering form the external environment had the goal to find or create a solid input/output interface. The communication phase relates to the communication between the Smart Object, the machine and an user. The final part, consists in the ”brain” of the SO. It is there where its behaviour is configured, where all the rules are processed in order to achieve the complete implementation of the SO. Finally some tests were conducted, by the creation of some default rules and using the inputs/outputs of the sensoring interface and the machine's PLC.

1.5 Structure

Apart from this introduction chapter, this dissertation is divided in the following chapters:

- **Chapter 2 - State of the Art**: Explanation of the ADVENTURE European Project, as well as all the background work and current developments made in the field of the Internet of Things and Smart Objects;
- **Chapter 3 - Smart Object for Manufacturing**: Presentation of our proposal of a Smart Object for Manufacturing. Later on, all the components of this SO will be exposed and discussed separately;
- **Chapter 4 - Solution Design**: Description of the solution design for the Smart Object by developing an I/O Interface, crucial to the data collecting on the surrounding environment;
- **Chapter 5 - Communication**: Study and implementation of different types of communications between distinct technologies and its interrelation to compose the Smart Object;
- **Chapter 6 - Processing**: Exposition of the processing part of the Smart Object, its ”Brain”. Its is divided in four sections, being the first two the presentation of the programming language and the database used. The last two sections concern the two different Python programs used for the rule interpretation task;
- **Chapter 7 - Results**: Discussion of the results of the several tests conducted to exemplify the behaviour of the Smart Object;

- **Chapter 8 - Conclusions and Future Work**: Final conclusions and achievement of goals, as well as future works of this project;
- **Appendix A - ADVENTURE**: ADVENTURE user interface example;
- **Appendix B - I/O Interface Schematics**: Compete schematics of the solution's PCB;
- **Appendix C - Code**: Full code of the communication protocol and both python scripts for the processing component;
- **Appendix D - User Visual Interface**: Concept of a possible future user visual interface.

Chapter 2

State of the Art

In this chapter it is explained the ADVENTURE European Project, as well as all the background work and current developments made in the field of the Internet of Things and Smart Objects.

2.1 ADVENTURE Project

ADaptive Virtual ENTerprise ManufacTURing Environment (Adventure) is a Small or Medium-Scale Focused Research Project (STREP) funded by the European Seventh Framework Programme in Virtual Factories and Enterprises. The goal of the project is the creation of a framework that provides the tools to combine factories in a pluggable way to manufacture a particular product. This includes the creation of manufacturing processes, finding partners as well as real-time monitoring of the processes that are put into play.

The concept of combining the power of several independent factories to achieve complex manufacturing processes as so-called virtual factories is not new and has been addressed by several research projects in recent years. However, most of them are limited to create virtual factories at a business level and in many cases they concentrate on the partner-finding and factory-building processes. Still, no proven tools and technologies exist in the market to provide valuable end-to-end integrated Information and Communication Technology (ICT) in such environments.

ADVENTURE will help virtual factories and enterprises move beyond existing operational limitations by providing concrete tools and approaches for leveraging the information exchange between factories. Factory process optimization will be enabled by the integration of runtime factory selection, forecasting, monitoring, and on-the-fly collaboration.

ADVENTURE aims at simplifying the establishment, management, adaptation, and monitoring of dynamic manufacturing processes in virtual factories by building on concepts and methods from the field of Service-oriented Computing and therefore benefiting from the progress that has been made in this domain over the last few years. Technologies from the field of Ubiquitous Computing and the Internet of Things, e.g., wireless sensors, will be adopted in order to support the monitoring and governance of processes, i.e., give information about the current status of manufacturing and delivery.[4]

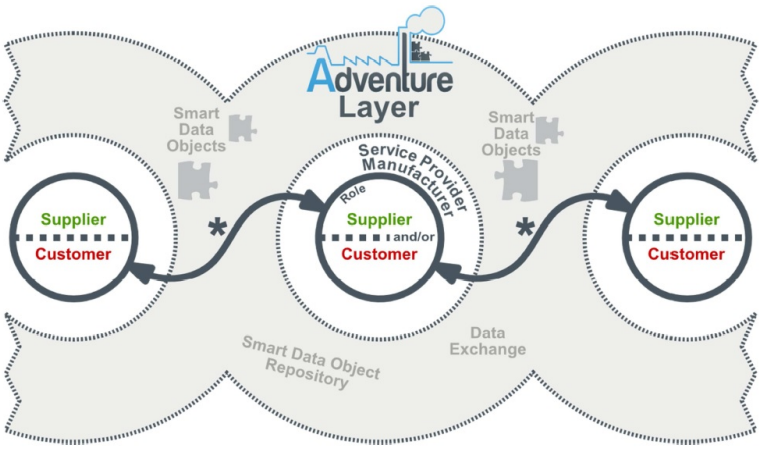


Figure 2.1: The ADVENTURE Layer [5]

Although this project will only develop the Smart Object framework, in appendix A we can see how the final ADVENTURE interface should be and the functionalities that must have.

2.2 Internet of Things

Over the last years a new paradigm as developed that will change everything, including ourselves, as we know nowadays – the Internet of Things (IoT). This may seem an audacious statement but if consider the impact of the Internet already has on education, business, science, humanity, etc. No doubt, ”the Internet is one of the most important and powerful creations in all of human history”. If we consider the IoT the evolution of the Internet, in the way that allows us to gather, analyse and distribute data that we can process and transform into information, knowledge and wisdom, IoT becomes extremely important.

2.2.1 Definition

The IoT is a computing concept that describes a future where everyday objects will be connected to the Internet and capable of identifying themselves to other devices. The term initially was related to RFID as the method of communication, but now it may include sensor and wireless technologies or QR codes. It allows an object to represent itself digitally, and no longer does the object relate to just us, he is now connected to other surrounding objects and data.

The Internet of Things is a difficult concept to define precisely. Although it exists many different definition of the term, they all share the idea that the first version of the Internet was all about data created by people, while the next version is about data created by things.

The Cisco Internet Business Solutions Group (IBSG) defines the IoT as "the point in time when more "things or objects" were connected to the Internet than people." [6]

Figure 2.2: Internet of Things

2.2.2 Concept

2.2.2.1 The birth of IoT

IoT's origins can be traced to the Massachusetts Institute of Technology (MIT), from work on the Auto-ID Center. Founded in 1999, this department was working in the field of networked radio frequency identification (RFID) and innovative sensing technologies. Kevin Ashton first mentioned the "Internet of Things" term in a presentation he made to Procter & Gamble. Here's how Ashton explains the potential of the Internet of Things:

> "Today computers – and, therefore, the Internet – are almost wholly dependent on human beings for information. Nearly all of the roughly 50 petabytes (a petabyte is 1,024 terabytes) of data available on the Internet were first captured and created by human beings by typing, pressing a record button, taking a digital picture or scanning a bar code. The problem is, people have limited time, attention and accuracy – all of which means they are not very good at capturing data about things in the real world. If we had computers that knew everything there was to know about things – using data they gathered without any help from us – we would be able to track and count everything and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling and whether they were fresh or past their best." [7]

2.2.2.2 IoT today

In 2003, there were approximately 6.3 billion people living on Earth and 500 million devices connected to the Internet [1]. If we do the math it is easy to see that it makes 0.08 devices per person. IoT didn't exist in 2003 because, according on Cisco IBSG's definition, there were few devices, as smartphones were just being introduced. The "boom" in smartphones and tablets PCs increased the number of devices connected to the Internet to 12.5 billion in 2010, bringing the number of connected devices per person to 1.84 (world's human population = 6.8 billion).

In Figure 2.3 we can see the point where the ration of devices per person as surpassed 1. That can be considered the true "born" of the IoT.

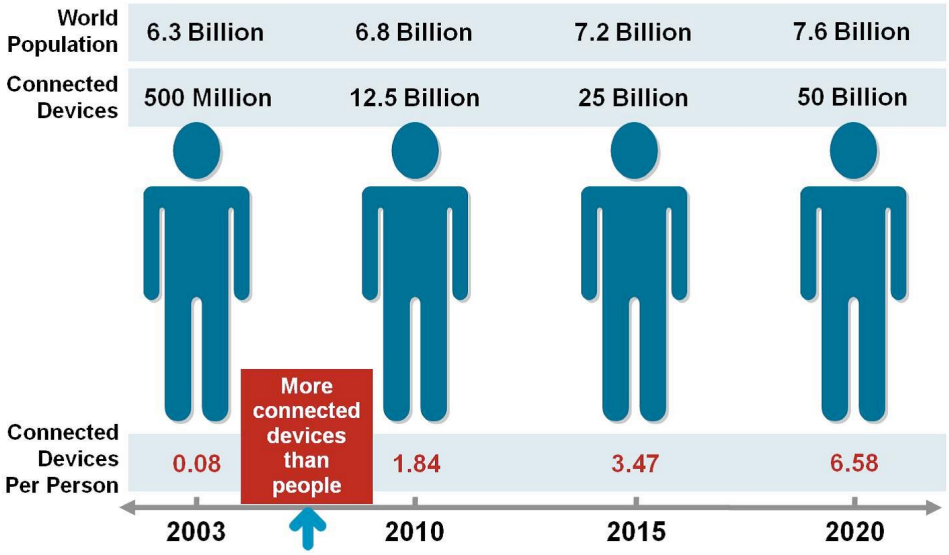


Figure 2.3: The IoT was "born" between 2008 and 2009 [6]

Today, IoT is well under way, as many initiatives as smart grids, intelligent vehicles and houses, networked business processes, etc. continue to progress and advance every day.

2.2.2.3 Trends

Internet of Things has been identified as one of the emerging technologies in IT as noted in Gartner's IT Hype Cycle (see Figure 2.4). A Hype Cycle represents the emergence, adoption and impact on applications of specific technologies. It has been predicted that IoT will take 5-10 years for market adoption. The popularity of these type of paradigms changes with time. The web search popularity, as measured by the Google search trends during the last 10 years for the terms Internet of Things, Wireless Sensor Networks and Ubiquitous Computing are shown in Figure 2.5. As it can be seen, since the appearance of the IoT, the search volume of the term as increased, as the trend for Wireless Sensor Networks as decreased. The dotted line at the end is the Google's search

[1] Sources: U.S. Census Bureau, 2010; Forrester Research, 2003.

forecast. It is predictable that the search volume will continue to increase as other technologies will converge to form a true and global Internet of Things. [8]

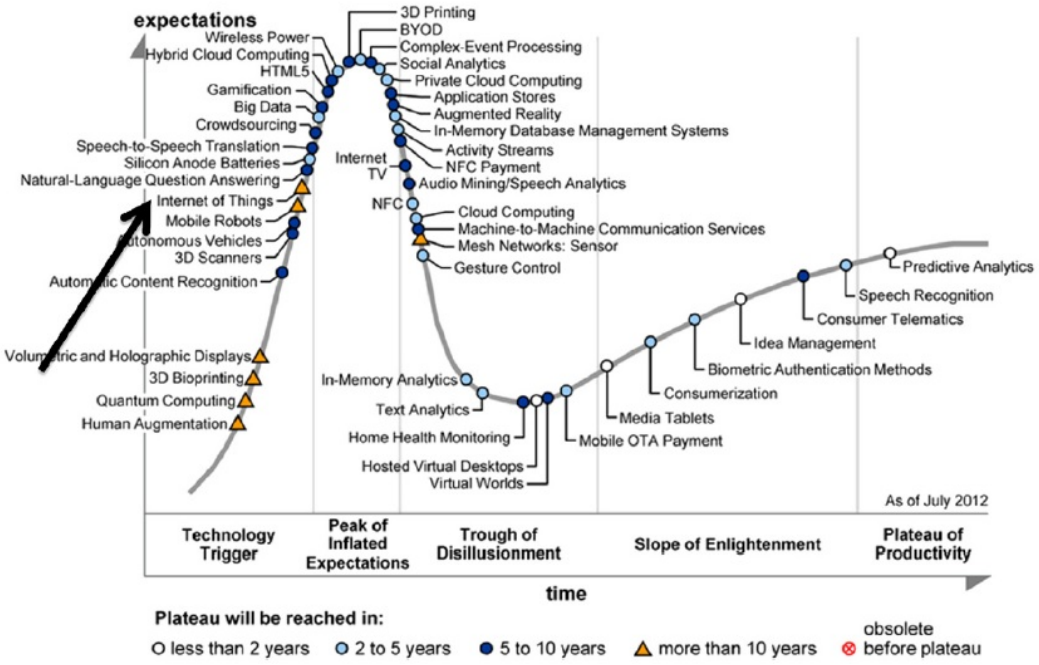


Figure 2.4: Gartner 2012 Hype Cycle of emerging technologies. [9]

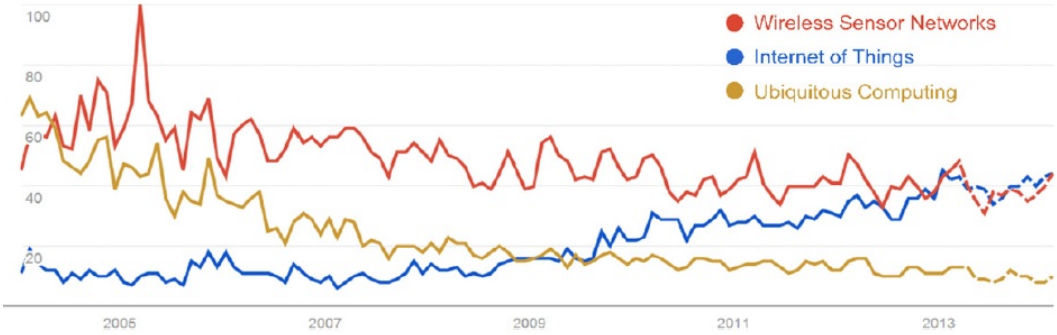


Figure 2.5: Google search trends since 2004 for terms Internet of Things, Wireless Sensor Networks, Ubiquitous Computing.

2.2.2.4 The importance of the IoT

The IoT is vital for Human Progression. "We evolve because we communicate" [6], this statement is essential for the understanding of IoT and its global goal. Once fire was discovered and shared it didn't need to be rediscovered, only communicated. There are infinite examples of this statement like the discovery of the helix structure of DNA that allowed the medicine and genetics disciplines to take giant leaps forward. The value of sharing information can be best understood be examining

how humans process data (see Figure 3). The pyramid layers are data, information, knowledge and wisdom. Data is the raw material processed into information. Volumes of data can help identify patterns or trends. Knowledge is information of which someone is aware. Wisdom is born from knowledge plus experience. While knowledge changes over time, wisdom is timeless.

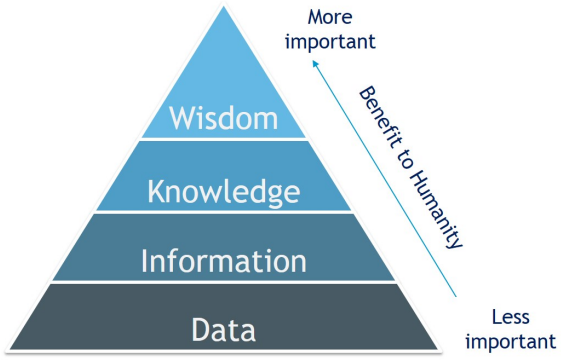


Figure 2.6: Humans Turn Data into Wisdom [6]

As we can see in Figure 2.6 there is a relation between the data (input) and wisdom (output). The more data is created and analysed, the more knowledge and wisdom people can obtain. IoT dramatically increases the amount of data available to process. This, together with the Internet capacity to communicate this data, will allow mankind to advance even further.

With the increase of the worlds population, it becomes more important for people to take care of Earth and its resources. Besides that, people want to live a healthy, fulfilling and comfortable life. Combining the ability of the IoT to sense, collect, transmit, analyse and distribute data on a worldwide scale with the way people process that information, humanity can have the knowledge and wisdom it needs to survive and prosper in the next decades and centuries.

2.2.2.5 Conceptual Architecture

We can divide the Internet of Things in three layers that are the base of the architecture of the concept:

- **Application;**
- **Network;**
- **Data-context;**

2.2.3 Applications

There are several applications of the IoT that can divided them two global categories:

- **Information and analysis**

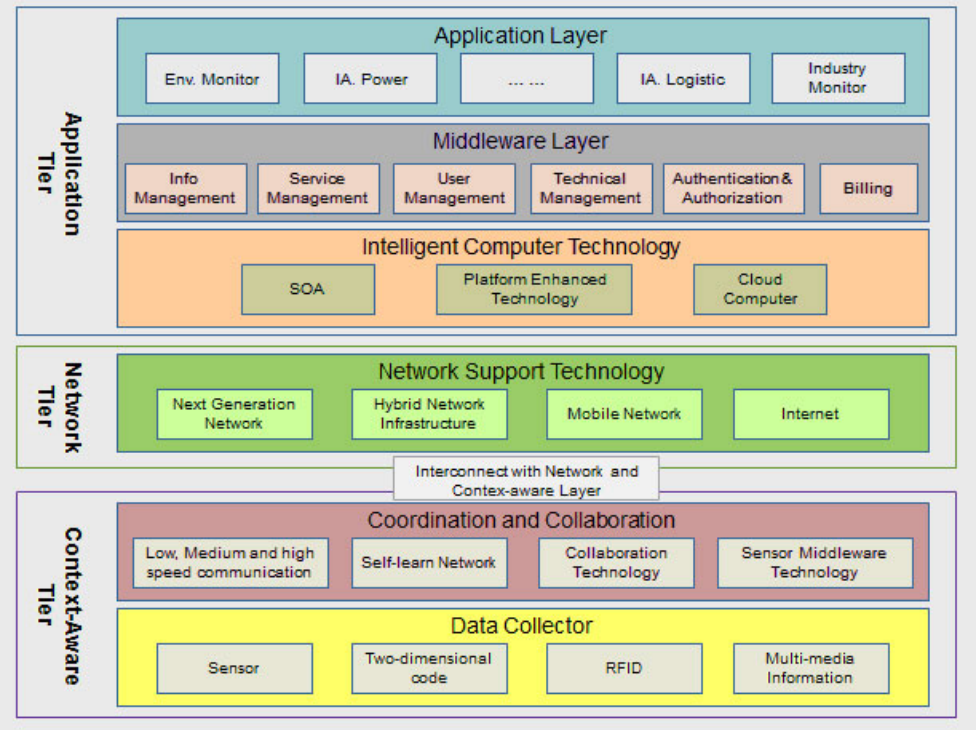


Figure 2.7: Conceptual Layers of the IoT [10]

- **Automation and control**

It is easy to note that both categories are related, in a way that we couldn't be able to control without the information and its posterior analysis.

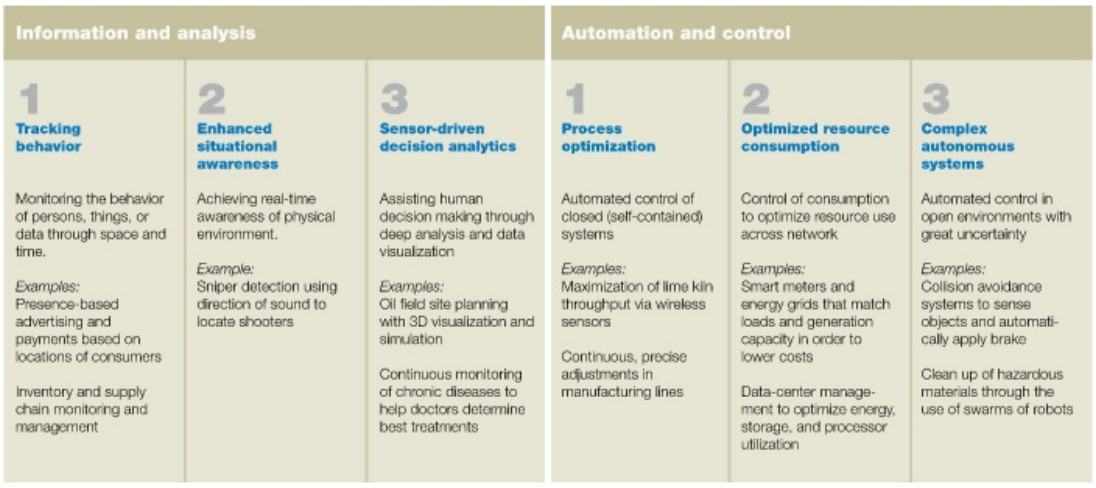


Figure 2.8: Applications of the IoT [11]

2.2.3.1 Real-world applications

The real world applications of the Internet of Things are enormous and probably never-ending. Since this is an area with constant development, there will never be a limit to the applications of IoT. Some of the "real-world" areas that IoT influences nowadays, according to [12], are:

- **Smart Cities** - benefits from parking spaces (monitoring of parking spaces availability in the city) and traffic congestion (monitoring of vehicles and pedestrian levels to optimize driving and walking routes) to smart Lighting (intelligent and weather adaptive lighting in street lights) and waste management (Detection of rubbish levels in containers to optimize the trash collection routes);

- **Smart Environment** - used in forest fire detection (monitoring of combustion gases and pre-emptive fire conditions to define alert zones) and air pollution (control of CO2 emissions of factories, pollution emitted by cars and toxic gases generated in farms);

- **Smart Water** - measure of water quality (Study of water suitability in rivers and the sea for fauna and eligibility for drinkable use) and water leakages 8Detection of liquid presence outside tanks and pressure variations along pipes);

- **Smart Metering** - used in smart grids (energy consumption monitoring and management), photovoltaic installations (monitoring and optimization of performance in solar energy plants) and tanks level (monitoring of water, oil and gas levels in storage tanks and cisterns);

- **Security & Emergencies** - applications in perimeter access control (access control to restricted areas and detection of people in non-authorized areas), liquid presence(liquid detection in data centers, warehouses and sensitive building grounds to prevent break downs and corrosion) or radiation levels control (distributed measurement of radiation levels in nuclear power stations surroundings to generate leakage alerts);

- **Retail** - used for supply chain control (monitoring of storage conditions along the supply chain and product tracking for traceability purposes), smart product management (control of rotation of products in shelves and warehouses to automate restocking processes) and in intelligent shopping applications (getting advices in the point of sale according to customer habits, preferences, presence of allergic components for them or expiring dates);

- **Logistics** - utilized in item location (search of individual items in big surfaces like warehouses or harbours) or to evaluate the quality of shipment conditions (monitoring of vibrations, strokes, container openings or cold chain maintenance for insurance purposes);

- **Industrial Control** - used in M2M applications (Machine auto-diagnosis and assets control), temperature monitoring (control of temperature inside industrial and medical fridges with sensitive merchandise) and Indoor Location (asset indoor location by using active (ZigBee) and passive tags (RFID/NFC)).

- **Domotic & Home Automation** - applications in energy and water use (energy and water supply consumption monitoring to obtain advice on how to save cost and resources), remote control appliances (switching on and off remotely appliances to avoid accidents and save energy) or intrusion detection systems (detection of windows and doors openings and violations to prevent intruders);
- **eHealth** - utilized for fall detection (assistance for elderly or disabled people living independent), sportsmen care (vital signs monitoring in high performance centers and fields or patients surveillance (monitoring of conditions of patients inside hospitals and in old people's home).

2.3 Smart Objects

The multiple technologies involved in the IoT concept represents one of the main challenges and requires an effective integration framework - a Smart Solution.

The basic concept behind the smart objects approach is to provide a single platform for the creation, processing and sharing of events based on sensor data, as well as a mechanism for accessing object conditions and their contextual networks via well defined, Internet-based interfaces. This section introduces the SO framework and its constitution for us.

2.3.1 Smart Solutions

The combination of a smart object and the service exploiting its capabilities is called a Smart Solution. Without it the SO would not be able to communicate with its surrounding environment, or retrieving data from it and, therefore, it would not be that smart.

2.3.1.1 User Benefits

In a way there is no novelty on smart solutions and objects. Telecommunication sets and consumer electronics devices are by nature smart objects, as they are fundamentally designed for connecting to a network such as the Internet. Indeed more than six billion smart objects in use today are personal computers (PC), smartphones and tablets. However, the real interest is in the vast number of other yet unconnected products. These can be divided in seven areas, as show in Figure 2.9.

Smart Objects enable consumers, businesses and communities to optimize and extend their operations by exchanging information through the network. Successful smart solutions could provide several benefits such as:

- Product lifetime extension. Continuous monitoring of a machine allows preventive maintenance, which increases machine lifecycle and productivity, for example by reducing downtime.
- Energy use optimization. By being used as a function of external conditions, devices can be made to consume less energy and/or run at a lower cost. For example, at sudden moments

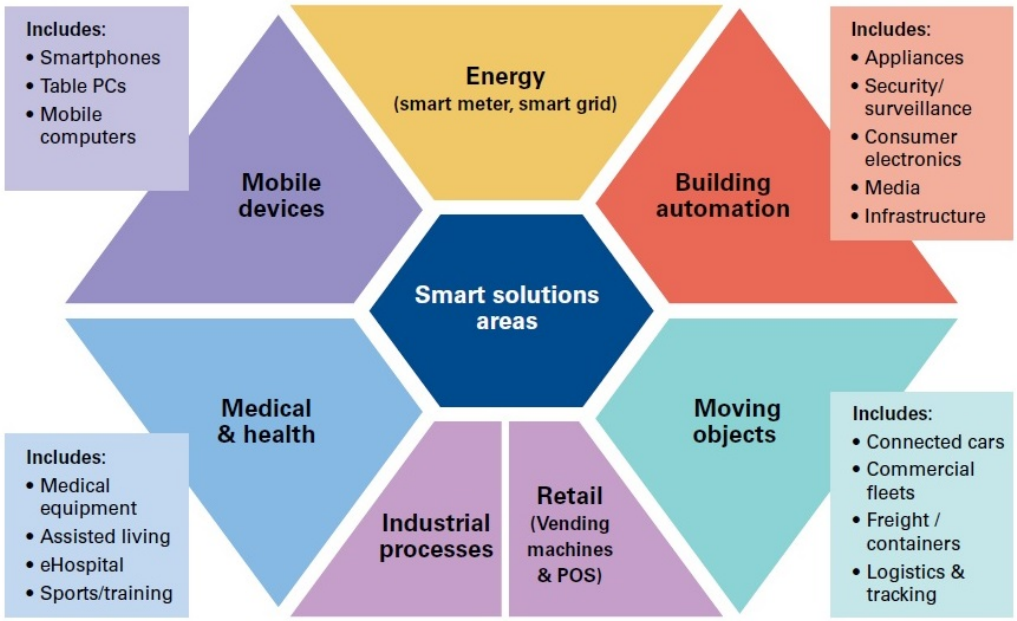


Figure 2.9: Areas with significant opportunities for Smart Solutions

of peak demand or supply shortage, a refrigerator could be switched off temporarily without risking its contents.

- Greater user convenience. Remote access to a product can improve its ease of use, such as in the case of patient care at home or remote power management.

- Provision of value-added services. The value of a product can be enhanced by tacking services onto it. For example, adding smart features to a car allows it to be tracked and recovered in case of theft.

Obviously, only smart solutions that create sizable benefits for both consumers and businesses will prevail .

2.3.1.2 Value Chain

Bringing smart solutions to life requires bringing together several physical components – such as the smart object – and services. As a consequence, the value chain is quite scattered and complex, as seen in Figure 2.10. As the propose of this dissertation is to idealize and implement a Smart Object we have to give a closer look at the value chain that it belongs. [13]

2.3.2 Definition

Smart Object are objects not only capable of providing their exclusive identification and condition, but also able to perform object-to-object communications, ad-hoc networking and object-centric

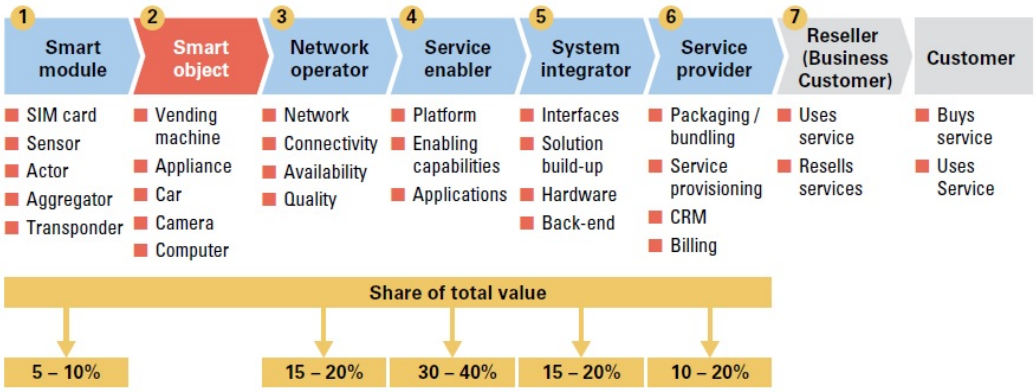


Figure 2.10: Participants in the value chain for Smart Solutions

complex decision-making. According to [14] the Smart Object is based on five fundamentals properties:

- Possess a unique identity;
- Are able to sense and store measurements made by sensor transducers associated with them;
- Are able to make their identification, sensor measurements and other attributes available to external entities such as other objects or systems;
- Can communicate with other Smart Objects;
- Can make decisions about themselves and their interactions with external entities.

Consumer goods, product parts, assembly machinery, logistics and transportation items and end-user resources can be monitored and provide valuable information to the ”outside”. To monitor their condition, we use embedded devices with wireless communication abilities. That device should be attached to the object, becoming a part of them, the same way as bar-code is part of almost all products nowadays.

2.3.3 Technology Background

Table 2.1 summarizes the most relevant work made, either on architectures that integrate RFID and sensors or on standardization efforts towards the realization of the IoT. As the table shows, although the technologies for collecting, processing and distributing information on objects already exist or are well advanced. There is almost none integration between them, leading to a lack of a complete platform for dissimilar data processing and sharing.

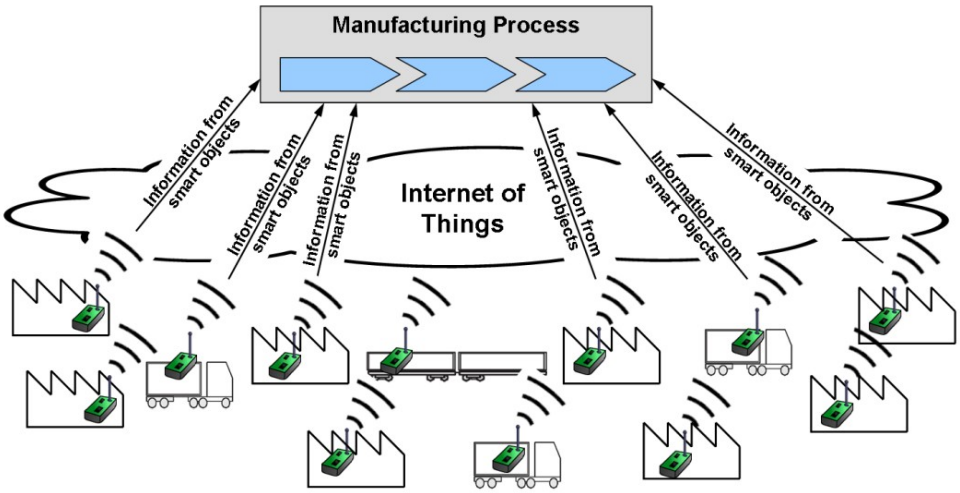


Figure 2.11: Smart Objects definition [5]

2.3.4 Components

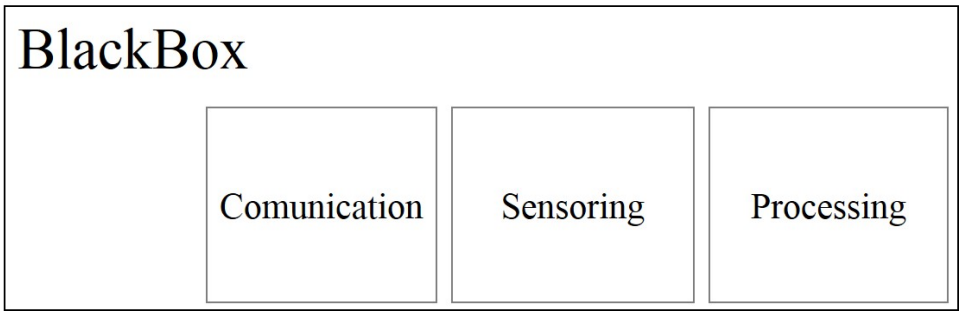


Figure 2.12: Smart Objects or "BlackBox" components

2.3.4.1 Sensoring

The sensoring part of the SO is our connection to what is physically happening. It is through them that we receive the data that we can later process. This sensoring can be made from different forms, from more simple solutions such as RFID sensor tags or Wireless Sensor Network to more complicated, e.g. an interface of inputs and outputs that can incorporate all solutions possible in just one single piece, as shown bellow.

Existing Solutions The first approach was to know what already exists in the market and if those suit this application. After some research there where several products identified as interesting, but that do not fit perfectly this application. From the study of all these products down, it was possible to idealize and build a proper product for this interface.

Table 2.1: Summary of RFID and Sensor related work [14]

	Description	Main shortcomings	Potential improvements
PROMISE [3]	EU project using Product Embedded Information Devices (PEID) for monitoring ID and condition of objects during their life cycle	Little alignment with standards, no networking of PEIDs	Adoption of ID standards. Consider sensor (object) networking
OGC SWE [4]	Extensive set of protocols and interfaces to share sensor information in a standard way over the Internet.	IDs not globally unique. Sensors are not considered to monitor objects or products	
EPC Network [1]	Emerging industrial RFID standards architecture based on unique item identification via the Electronic Product Code (EPC)	Does not yet handle sensor data	Extend current standards with sensor data
BRIDGE [5]	EU project for developing new technologies within the EPC Network	Work with sensors does not extend the EPC Network	
EPC SN [7]	Auto-ID lab project to extend the EPC Network with sensors	Too complex to allow a full architecture extension	Compromise in developing a simpler functional part of the extension
ISO 18000.6, 24753, IEEE 1451.7	Set of standards dealing with the integration of RFID with sensors	Under development. Cooperation among standardization bodies is complex and slow	Standards need to reach a mature state before they can be used
GSN [13]	Middleware to collect and share information from RFID and WSN over the Internet	RFID and WSN data is not integrated at object level	Provisions for collecting both RFID and sensor data from an object
Mitsugi et al. [14]	General architecture for managing RFID and WSN integrated information	No description on real-world implementations	Further detail is needed to realize their vision
SARIF [11]	Middleware for designing applications requiring RFID and WSN information	Integration is only by spatial comparison, no mention of Internet scalability	Consider more integration methods and an explicit connection with Internet protocols

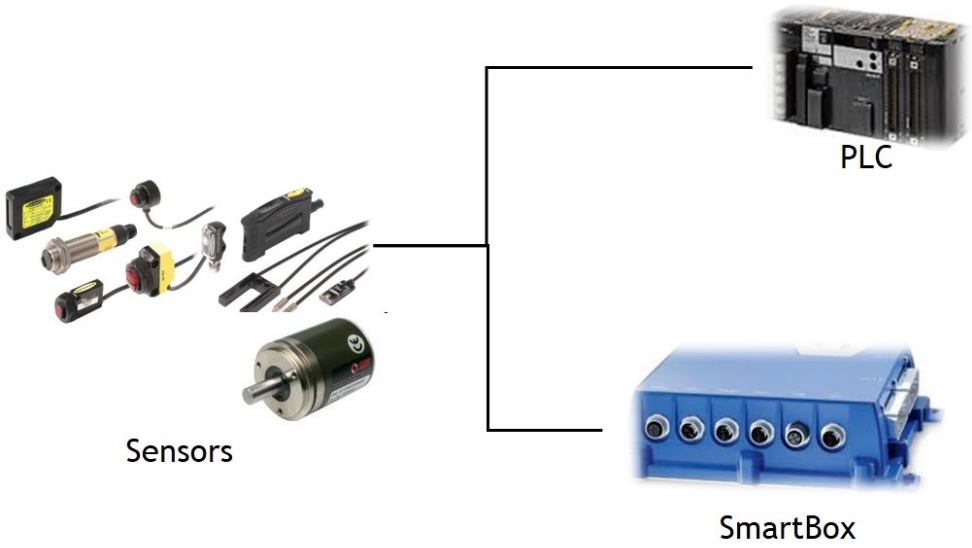


Figure 2.13: Sensoring interactions

1. OpenSprinkler - Beagle-Master[15]

1.1 Overview

"OpenSprinkler Beagle (OSBo) is an open-source sprinkler / irrigation extension board for the BeagleBone Black. It is based on the design of OpenSprinkler, but its 'brain' is a BeagleBone

instead of an AVR microcontroller. OSBo makes use of BeagleBone's GPIO pins to directly control sprinkler valves. By stacking a BeagleBone Black on top of OSBo, you instantly have a low-cost, web-connected sprinkler controller. OSBo is expandable — by linking OpenSprinkler Zone Expansion Boards, you can extend to an unlimited number of stations. You can also make use of the digital/analog pins of the BeagleBone Black to interface with external sensors and actuators."

1.2 Design

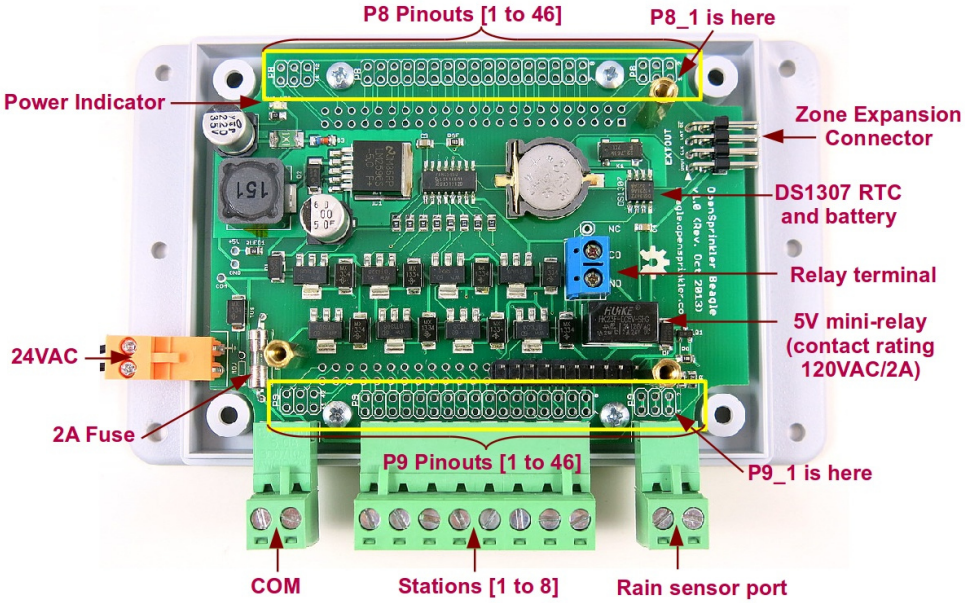


Figure 2.14: OpenSprinkler

1.3 Specifications

Table 2.2: Specifications of the OpenSprinkler - Beagle-Master

Technical Details	
Operating Voltage	22 to 28V AC
Output Current	650mA @ 5V to power BeagleBone Black
Built-in Components	24V AC to 5V DC converter, RTC with battery, rain sensor terminal, fuse, mini-relay, per-station TVS.
Maximum # of Zones	No software limit
Max. Current Per Zone	800mA continuous (@24VAC), 8A impulse / inrush
Package Inclues	One assembled and tested OSBo board, separation pillars, terminal blocks, and product case. (===DOES NOT=== include 24V AC sprinkler transformer or valves, which are available at Amazon.com or local home improvement stores). (===DOES NOT=== include Raspberry Pi).
Dimension and Weight	
Size	135mm x 85mm x 42mm (5.3" x 3.3" x 1.65")
Weight	170g (6.1oz)

2. Osso Beaglebone I/O expansion cape[16]

2.1 Product Description

Osso expand the BeagleBone board (both white or black) adding 8 relays and 8 opto-isolated digital inputs.

2.2 Features

- 8 relays up to 275V AC @10A
- 6 digital opto-isolated inputs up to 50 meters distance
- Power supply 12V DC (Beaglebone powered from expansion pins)

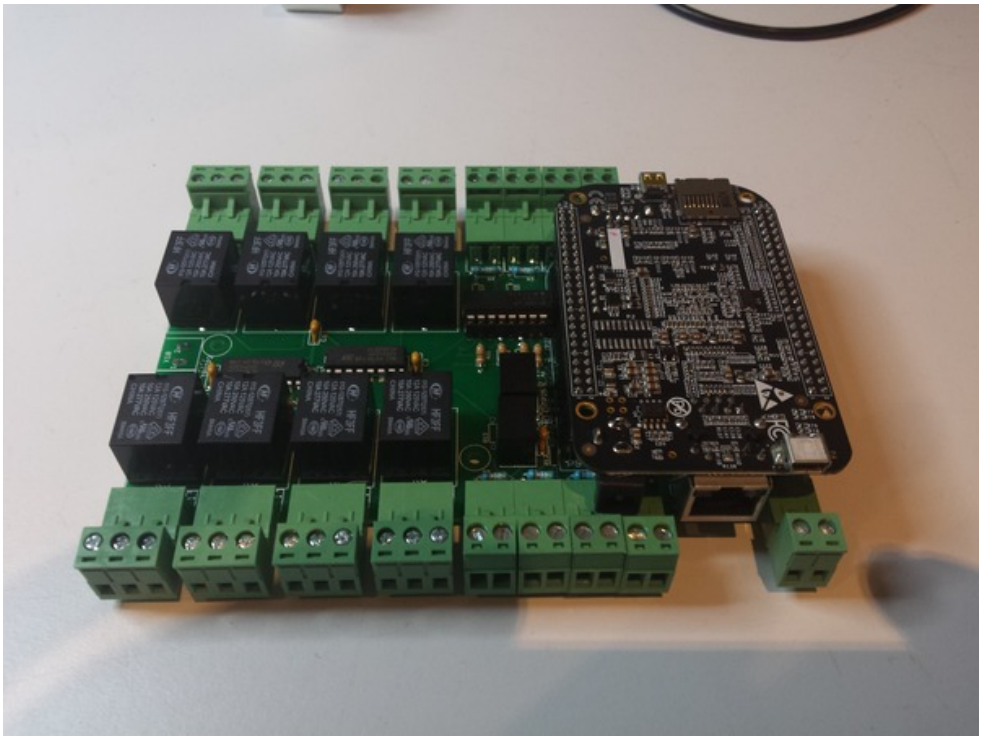


Figure 2.15: Osso I/O

3. MOD IO[17]

3.1 Overview

MOD-IO is a stackable development board which adds analog and digital inputs and outputs to any of our development boards with UEXT. If you work with any of our development boards with a UEXT connector and you need more digital and analog inputs and outputs, you can add these by connecting MOD-IO to your development board. This board allows easy interfacing to 4 relays, 4 optoisolated digital inputs, 4 analog inputs. MOD-IO is stackable and addressable and what does this mean? It means that these boards can plug together so that you can add as many

inputs and outputs as you want! The MOD-IO has an ATmega16 microcontroller and the firmware is available for modification.

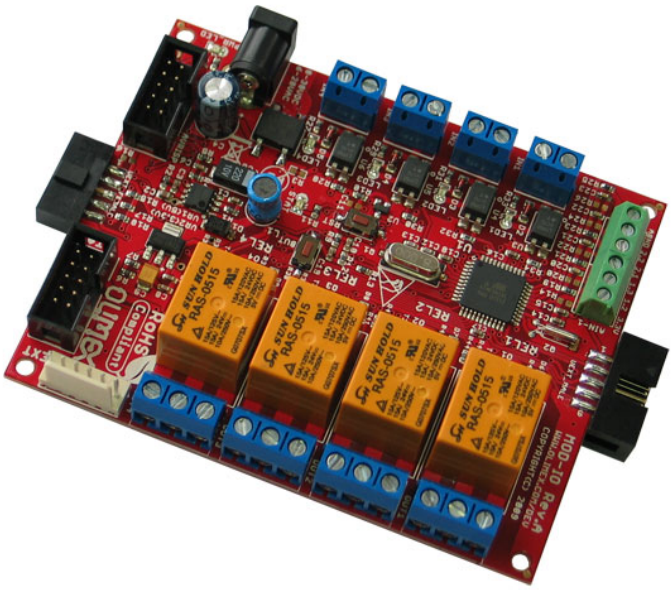


Figure 2.16: MOD-IO

3.2 Features

- Open source hardware board with ATMega16L-8AU microcontroller
- ICSP 5x2 pin connector for in-circuit programming with AVR-PG1, AVR-PG2, AVR-ISP500, AVR-ISP500-TINY, AVR-ISP500-ISO or other compatible to 10 pin ICSP layout
- JTAG 5x2 pin connector for in-circuit programming with AVR-JTAG, AVR-JTAG-USB or other compatible to 10 pin JTAG layout
- EXT extension connector for the unused AVR ports
- Status LED
- Reset IC ZM33064
- Quartz crystall oscilator circuit 8MHz
- DC-DC with input voltage 8-30VDC - this board can be powered from 24V industrial power supplies
- Power plug-in jack
- 4-optocoupler isolated inputs with screw terminals

- Input status LEDs
- 4-relay outputs with 5A/250VAC contacts with screw terminals
- Output status LEDs
- Four mounting holes 3.3 mm (0.13")
- FR-4, 1.5 mm (0.062"), red soldermask, white silkscreen component print
- Dimensions 80x100 mm (3.9 x 3.15")

2.3.4.2 Processing

This component is the ”brain” of the SO. It is responsible for interacting all its components and parts. Without this, it would be impossible to process the results of the sensoring or define the communication to use. Its versatility allows and defines multiple purposes for the SO and its environment.

Micro-processor

Nowadays exist numerous micro-processors, each one with some advantages and disadvantages. Between those I have selected the BeagleBone Black® (BBB), Figure 2.17 and the Raspberry PI® (RPI), Figure 2.18.

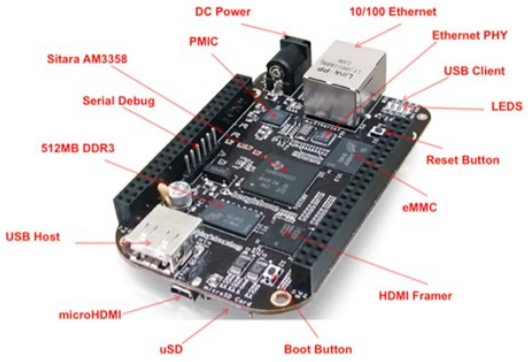


Figure 2.17: BeagleBone Black

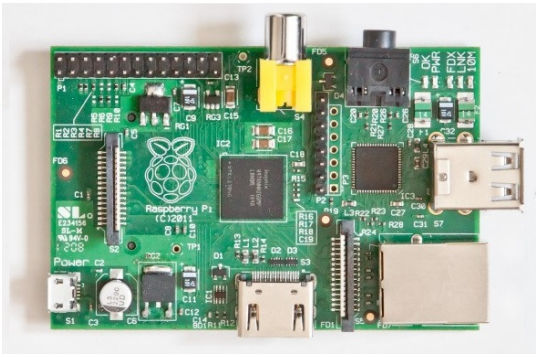
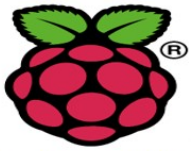


Figure 2.18: Raspberry PI

Comparing both products, as shown on Figure 2.19, and concerning only the features needed for this project, although the BBB is a more expensive micro-processor and consumes a bit more energy, it has a more capable processor and more important it has 65 GPIO pins, much more than the 8 pins of the RPI. This makes the BBB the right choice for this project.



Comparing Raspberry Pi and BeagleBone Black

	BeagleBone Black	Raspberry Pi
Base Price	45	35
Processor	1GHz TI Sitara AM3359 ARM Cortex A8	700 MHz ARM1176JZFS
RAM	512 MB DDR3L @ 400 MHz	512 MB SDRAM @ 400 MHz
Storage	2 GB on-board eMMC, MicroSD	SD
Video Connections	1 Micro-HDMI	1 HDMI, 1 Composite
Supported Resolutions	1280×1024 (5:4), 1024×768 (4:3), 1280×720 (16:9), 1440×900 (16:10) all at 16 bit	Extensive from 640×350 up to 1920×1200, this includes 1080p
Audio	Stereo over HDMI	Stereo over HDMI, Stereo from 3.5 mm jack
Operating Systems	Angstrom (Default), Ubuntu, Android, ArchLinux, Gentoo, Minix, RISC OS, others...	Raspbian (Recommended), Ubuntu, Android, ArchLinux, FreeBSD, Fedora, RISC OS, others...
Power Draw	210-460 mA @ 5V under varying conditions	150-350 mA @ 5V under varying conditions
GPIO Capability	65 Pins	8 Pins
Peripherals	1 USB Host, 1 Mini-USB Client, 1 10/100 Mbps Ethernet	2 USB Hosts, 1 Micro-USB Power, 1 10/100 Mbps Ethernet, RPi camera connector

Figure 2.19: BeagleBone Black vs Raspberry PI [18]

2.3.4.3 Communication

On what concerns to SO, there are two types of communications: between the SO and the machine/sensors (machine oriented) and between the SO and the user interface (world oriented).

The communication between the SmartObject and the Interface with the user can be made using several technologies: RFID, Wifi, Ethernet, USB, etc. For this thesis the main technology is Wifi (IEEE 802.11).

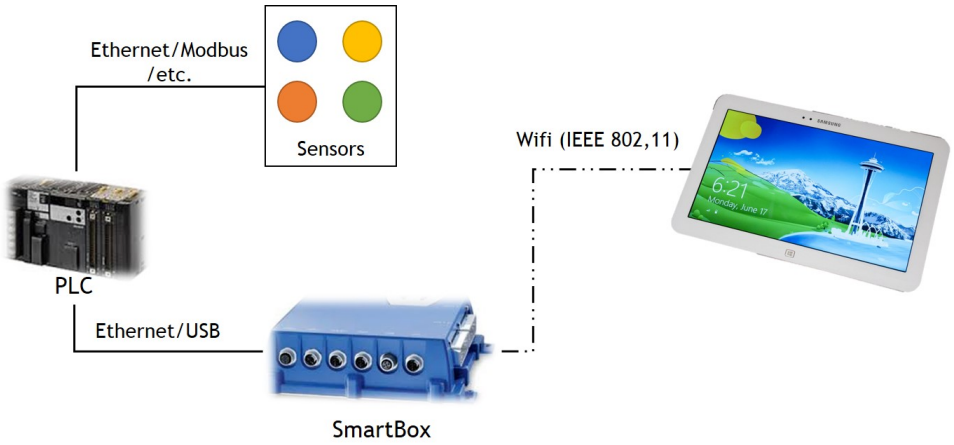


Figure 2.20: Main communications envolving the SmartBox

2.3.5 Existing Products

2.3.5.1 Wapice® Remote Control and Management

Currently Wapice® has developed a number of technology solutions for remote control and management, on other words, a Smart Object. Their solutions can work as part of a project or assembled in a full-scale application. They created a complete remote management system, as we can see in Figure 2.21, the WRM® - a complete remote control and management system with a wide range of features which includes the electronics, server and software. An overview of this product is available on Figure 2.22

Figure 2.21: WRM® 365

Features include:

- Servers that can be located anywhere; SaaS also possible
- Supported operating systems: Windows Server, Linux, UNIX
- Java EE Application Server (BEA WebLogic, IBM Websphere, JBoss)
- Database platforms (Oracle, MS SQL Server, IBM DB2, My SQL)
- Intuitive user interfaces
- Wireless, wired and satellite-based communication solutions
- Different views for different user profiles
- Enterprise Application Integration (EAI): WebService, JMS, RDB

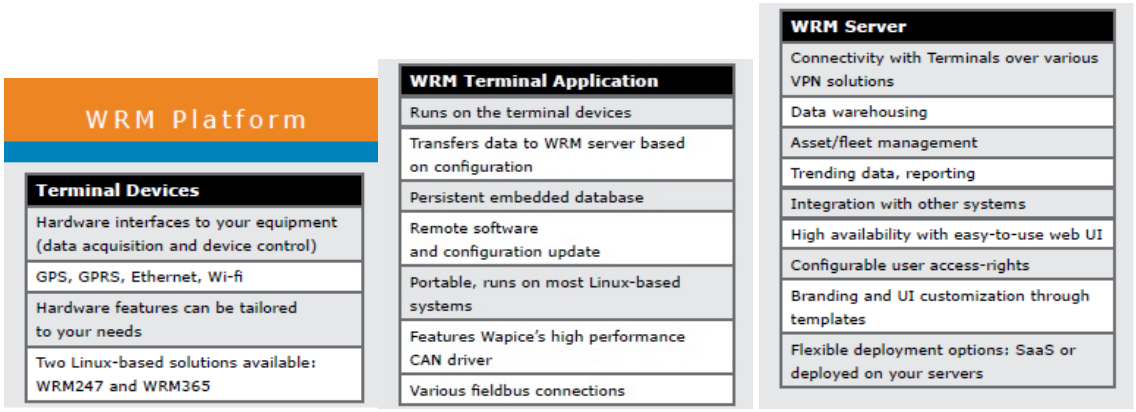


Figure 2.22: WRM overview [19]

Chapter 3

Smart Object for Manufacturing

In this chapter its presented our proposal of a Smart Object for Manufacturing. Later on, all the components of this SO will be exposed and discussed separately.

What do we want this Smart Object to do? This is the question that we have to answer in order to do the concept, requirements and functionalities

3.1 Concept

This Smart Object as to be able to collect data from its surrounding environment and to process that data as the user intends. It is now possible to identify three major components of the SO:

- Sensoring;
- Communication;
- Processing.

The sensoring part will be responsible for collecting information from the SO exterior, the communication used for interaction with the PLC on the operating machine (data gathering from the PLC) and the processing part, the ”brain” in this SO, responsible for connecting and interpret the data gathered from both parts. Each one of these components will have its own chapter to be further analysed.

This SO has to be capable of integrate a complex framework, such as the ADVENTURE Project. In Figure 3.1 we can note the importance of Smart Object in the project and how it is related to the other components.

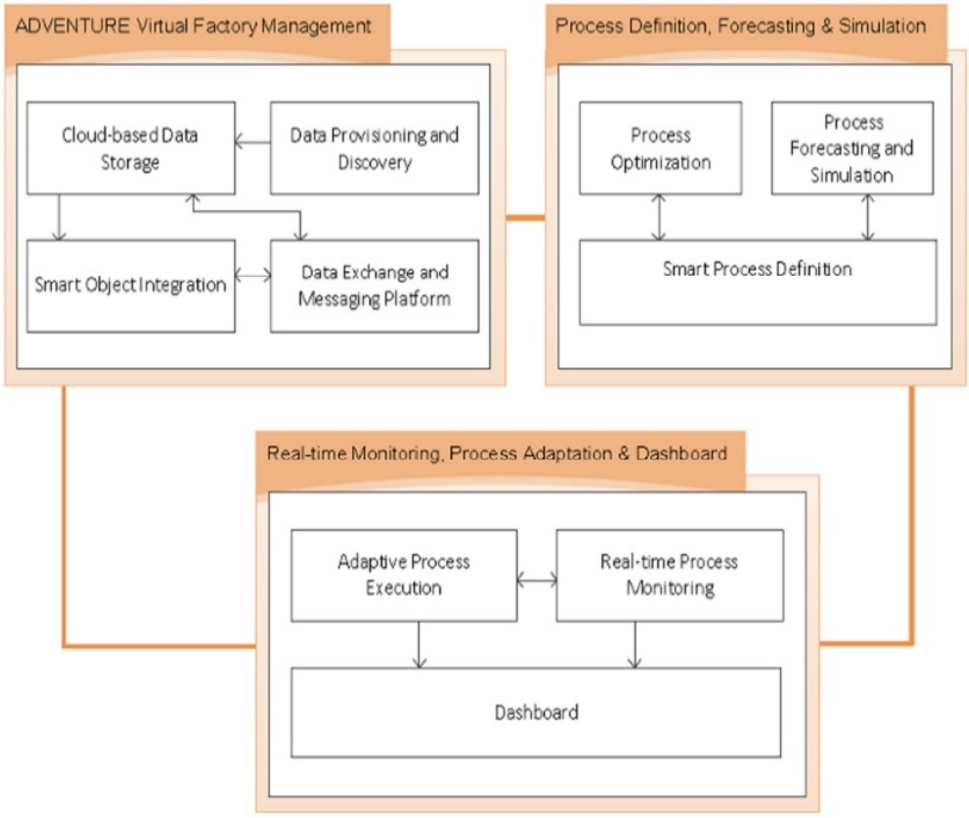


Figure 3.1: ADVENTURE components

3.2 Requirements

It is important to consider the requirements of the SO so that, when this project is finished, we can analyse which of them are complete and operational. This also gives a solid base to build the SO, part by part, and accomplish all goals previously purposed for this project. We can divide them in functional and non-functional requirements:

- **Functional**

  · **Behaviour based on events;**

  Description: The performance of the SO is defined by the external and internal events detected by the object. This defines the rule processing of the solution.
  Input: External and internal inputs or algorithms pre-configured.
  Output: Actions based on the rules veracity.

  · **Industrial implementation platform;**

  Description: The SO must be prepared for implementation in every industrial environment and every type of machinery. Compatible interfaces and communications must be implemented.

Input: Industry conditions and limitations.

Output: Smart Object ready for implementation and operation.

· **Communicate with external information sources;**

Description: Communication with other machinery components, such as PLC or input/output cards.

Input: PLC and cards data.

Output: Data storage in the database.

· **Behaviour configuration.**

Description: The user can configure and program the Smart Object to and for what is necessary.

Input: User needs.

Output: Smart Object configured and ready.

- **Non-functional**

· **Start-up ready;**

Description: The BBB is operational when powered up.

· **High processing capacity;**

Description: This is an important requirement because the fastest the processing cycle, the fastest the inputs are actualized and processed.

· **Compatible with different technologies;**

Description: The solution must be compatible with several types of micro-processor and PLC integration.

· **Low energy consumption;**

Description: Energy efficiency is very important these days in every type of industry and this SO can not consume a considered amount of energy.

· **Reduced Size**

Description: As this solutions intents to be ”invisble to the eye”, the smaller the SO, the easier its implementation and less space occupies in the machine.

3.3 Functionalities

Considering the functional requirements defined in 3.2, the functionalities that the SO must have are:

- **Acquiring information from sensors**
  The solution is able to collect data from sensors located in the surrounding environment and to process that data on-time.

- **Access to external information (through communication mechanisms)**
  By connecting the solution to a PLC or a input card, this solution is able to access the input/output memory area of those components.

- **Processing on the rules engine**
  Allowing the creation, editing and removing of the rules, the solutions is capable of processing those rules, using the data collected from the inputs and evaluate his veracity.

- **Configuration of the SO engine, according to different needs**
  As every case is unique, different needs of utilization can be configured in the SO, so that the system does what requested from the user.

- **Generation of actions**
  As result of the veracity of the rules, this solution is capable of generating some actions, either it is sending an email or actuate a relay controlled output.

3.4 Rules

The behaviour of this SO is implemented and configured by the execution of rules and their respective actions. These rules are the key to the autonomy and ”decision-making” of the Smart Object and they were divided in four main categories: discrete, continuous, internal and extra. For each rule, an action is executed if the rule is verified.

3.4.1 Discrete Rules

This type of rule concerns events or external discrete variables/inputs, such the digital and PLC inputs. Some of the discrete rules for this concept are:

- **Events rising edge (RE) or falling edge (FE);**
  Parameters 1st level: -
  Parameters 2nd level: -
  Description: When a change from 0 to 1 (RE) or from 1 to 0 (FE) in Var 1 detected, action happens.
  Example: If the rated current of the main motor exceeds the limit set, should issue an information message level A.

- **Events counter;**
  Parameters 1st level: X
  Parameters 2nd level: -
  Description: When Var 1 is bigger/lower/equal than X, action happens.
  Example: If the rated current of the main motor exceeds the limit set, should issue an information message level A.

- **Events period;**
  Parameters 1st level: X
  Parameters 2nd level: -
  Description: When Var 1 execution period is bigger/lower/equal than X, action happens.

Example: If a position sensor is on for a period longer than the limit set, an action happens.

- **Events counter in time-window;**
  Parameters 1st level: X
  Parameters 2nd level: Y
  Description: When Var 1 is greater than X in Y time window , action happens.
  Example: If motor circuit breaker fires over 4 times in a 2 hours time window, send e-mail to supervisor.

3.4.2 Continuous Rules

There are several inputs that can not be inserted in discrete events - continuous inputs. These inputs are characterized by having a wave form and some of those rules considered for the Smart Object are:

- **Reach threshold;**
  Parameters 1st level: X
  Parameters 2nd level: -
  Description: When Signal 1 exceeds or it is below X, action happens.
  Example: If a continuous and volatile motor voltage signal exceeds the set value, an alarm occurs.

- **Cross threshold limits;**
  Parameters 1st level: X
  Parameters 2nd level: Y
  Description: When Signal 1 misses the interval created by X and Y, an action happens.
  Example: If the signal of current in a motor is not in the interval previously defined, an alarm occurs.

- **Variable pattern change;**
  Parameters 1st level: Signal X
  Parameters 2nd level:
  Description: When Signal 1, in a time interval, escapes the pattern given in Signal X, an action happens.
  Example: The signal of a current in a motor as to respect a certain pattern, when not an e-mail is sent to the supervisor.

3.4.3 Internal Rules

Like the continuous rules, these ones concern continuous values, except they are not for external inputs, but from the Smart Object itself. They constitute some safety rules for the proper functioning of the SO. Some of those rules are:

- **Current value modification;**
  Parameters 1st level: X
  Parameters 2nd level: -
  Description: When a internal current value exceeds the expected value, an action happens.
  Example: If the current in a component of the SO exceeds the maximum value, an alarm occurs.

- **Reach threshold;**
  Parameters 1st level: X
  Parameters 2nd level: -
  Description: When Signal 2 (internal) exceeds or hits values below the limit, an action happens.
  Example: If the current output signal exceeds the maximum value for the component connected to that output, is it immediately deactivated.

- **Gradient surpass;**
  Parameters 1st level: -
  Parameters 2nd level: -
  Description: When the gradient of an internal Signal 2 is not what expected (positive or negative), an action happens.
  Example: If the output current of an output, which is expected to increase (positive gradient), decreases (negative gradient), that output is deactivated.

3.4.4 Extra Rules

Apart from the previous rules, there are what can be considerated ”extra” rules or functions. This meaning, rules that will increase the innovation and complexity of the Smart Object. These rules, for being much more complex are only described being its study and implementation a possible future work. Some of those rules are:

- **Neural Networks;**
  Description: Create an implement artificial neural networks for machinery learning and pattern recognition.

- **Overall Equipment Effectiveness (OEE);**
  Description: This function allows the measurement of the OEE, in the machine were the SO is installed.

- **High-Pass or Low-Pass Filters;**
  Description: Enables the creation of a high-pass or low-pass filters to an input continuous

variable, for example, to reduce the noise on an input.

3.4.4.1 Actions

- **Activate Output;**
  Parameters: X
  Description: Activates Output X.

- **Alarm;**
  Parameters: X
  Description: Activates an alarm during X seconds.

- **Send e-mail;**
  Parameters: x@gmail.com, y@gmail.com, Text
  Description: Send an email from x@gmail.com, to y@gmail.com containing the message ”Text”.

Chapter 4

Solution Design

This chapter covers the design of the solution for the Smart Object by developing an I/O Interface, crucial to the data collecting on the surrounding environment.

4.1 Concept

In order to allow the Smart Object(SO) and its ”brain” (the BeagleBone Black (BBB)) to interact with the surrounding industrial environment, it was suggested the idealization of an input/output interface (in form of a Printed Circuit Board), along with some crucial components to the proper functioning of the system.

Initially there were defined the key features that this component should have:

- Digital inputs
- Analog inputs
- Outputs

Hereupon the initial concept scratch was as shown in Figure 4.1.

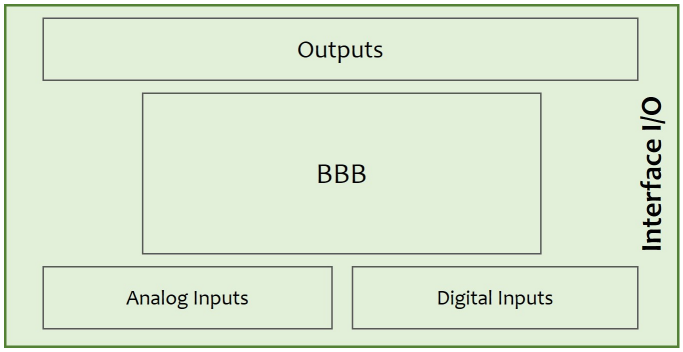


Figure 4.1: Concept Diagram

4.1.1 Digital Inputs

This inputs that should be used for connecting binary sensors, buttons and all kind of on/off switches. It is very important that the input and the output(input of the BBB) are two isolated circuits, so that we can protect the BBB from high voltages on the input. The GPIO pin supports a maximum voltage of 3.3V [20], so this circuit must be projected to a maximum 3.3V output whatever the input is.

4.1.2 Analog Inputs

The analogical inputs should work with both voltage and current supply. As the maximum voltage in the BBB Analog-to-digital converter (ADC) is 1.8V [20] there has to exist a voltage divider and a resistor that can provide the 1.8V maximum in the circuit output. To allow the possibility of voltage/current supply there will exist jumpers, so that the user can choose what type of supply suits him the best.

4.1.3 Outputs

As the main purpose of the Smart Object is to interpret the rules existing in its ”brain”, it is necessary that something demonstrates that the rule was fulfilled. This result of the veracity of the rules are the outputs for each rule that can go to a email sending to a LED light, or whatever the user wants, connected to the physical outputs of the interface.

4.2 Production

In order to go from concept to production of the Printed Circuit Board (PCB), was drawn an initial ”sketch” of the layout, illustrated on Figure 4.7, to start the design of the final schematic.

4.2.1 Digital Inputs

Since the main goal was the independence of the BBB input and the interface input, the use of an optocoupler on the circuit is crucial. It transfers electrical signals between two isolated circuits by using light and therefore protecting the system that receives the signal from high voltages.[21]

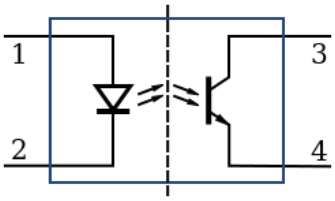


Figure 4.2: Schematic diagram of an opto-isolator showing source of light (LED) on the left, dielectric barrier in the center, and sensor (phototransistor) on the right.[1]

Along with the right components (a resistance and a diode), the optocoupler results in the correct circuit for each digital input as shown in Figure 4.3).[17]

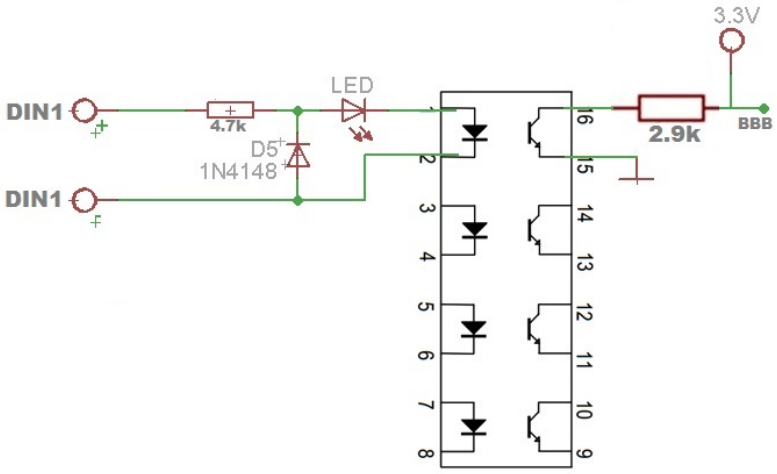


Figure 4.3: Digital Input

4.2.2 Analog Inputs

As the maximum voltage of the BBB ADC is 1.8V and the maximum input on the interface is 10V/20mA, there has to exist two circuits: one voltage divider for 10V maximum and one resistor for the 20mA current maximum.

Voltage Input

In this case a simple voltage divider is enough to transform the input of 0-10V into an output of 0-1.8V. According to (4.1) if $R_2 = 1k\Omega$, then $R_1 = 4.55k\Omega$. The result is the 3.3V has the image 4.4 demonstrates.

$$\begin{aligned} V_o &= \frac{R_2}{R_1 + R_2} \cdot V_i \\ \Leftrightarrow 1.8 &= \frac{R_2}{R_1 + R_2} \cdot 10 \\ \Leftrightarrow \frac{R_1}{R_2} &= 4.55 \end{aligned} \tag{4.1}$$

[1]Real-world schematic diagrams omit the barrier symbol, and use a single set of directional arrows.

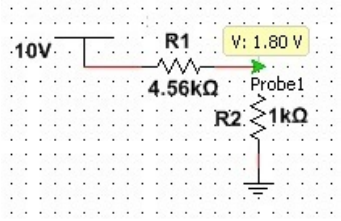


Figure 4.4: Simulation of the Analogic Voltage Input

Current Input

To be able to output a 1.8V from a maximum current of 20mA it is just simple as using the Ohm's Law. From the equation (4.2), if we use a 90Ω resistor, we have a 1.8V voltage has the Figure 4.5 shows.

$$\begin{aligned} R &= \frac{V}{I} \\ \Leftrightarrow R &= \frac{1.8}{20m} \\ \Leftrightarrow R &= 90\Omega \end{aligned} \tag{4.2}$$

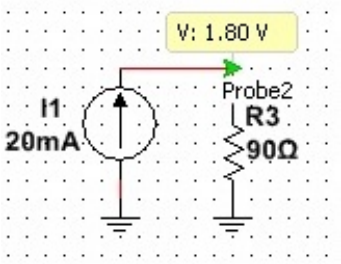


Figure 4.5: Simulation of the Analogic Current Input

4.2.3 Relay Controlled Outputs

The other essential component of the interface is the outputs. This outputs are the result of the rule, in other words they represent the action associated with a rule, activated when the condition is verified. Since the BBB provides 3.3V of direct output and the ideal is to give the user the possibility to work with any voltage, it is used a relay activated with a high voltage, high current darlington array, in this case a ULN2003 [22]. The Figure 4.6 displays the four outputs from the BBB, through the ULN2003 and then triggering the relay.

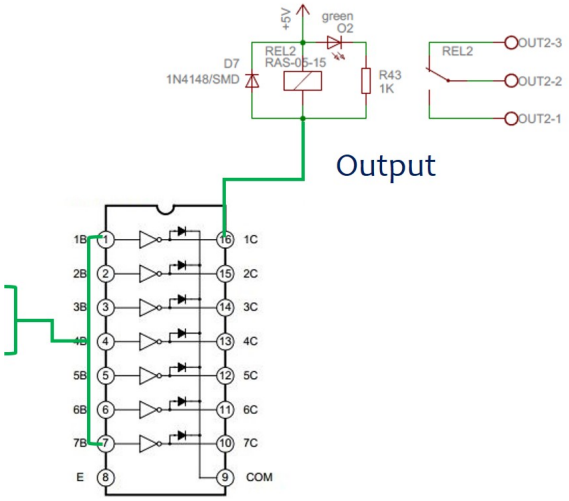


Figure 4.6: Relay Output with ULN2003

The draft at 4.7 allowed designing the accurate layout of the interface and the essential improvement of the scheme. It was now obvious and mandatory that this interface should have more features than those proposed before, therefore there were added three more features to it:

- Real-time Clock
- Serial Port
- DC-DC Converter (24V to 5V)

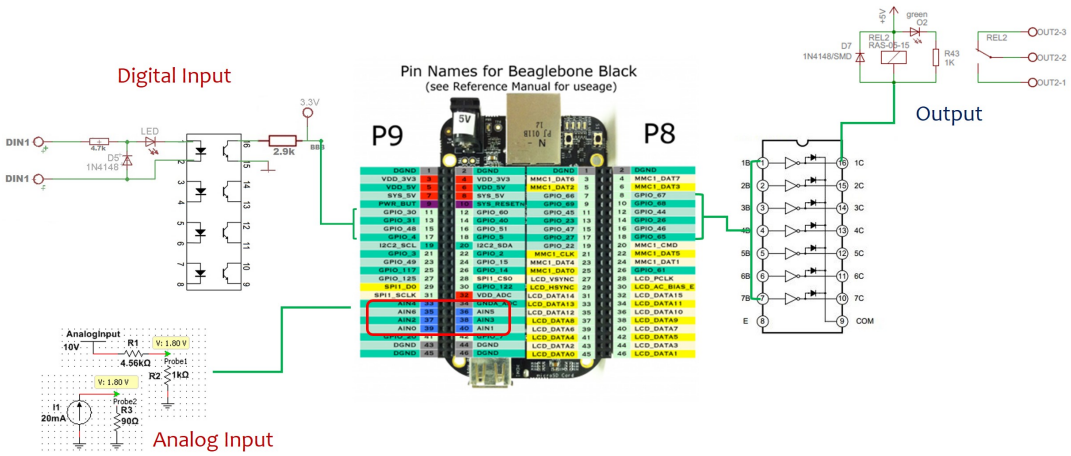


Figure 4.7: Interface I/O-Beta version

4.2.4 Real-time Clock

Seeing that the internal clock of the BBB could not keep the correct date when disconnected, it was necessary to implement a Real-time clock (RTC) on the board. It was used the schematic of the OpenSprinkler RTC [15], using a DS1307 as displayed in Figure 4.8

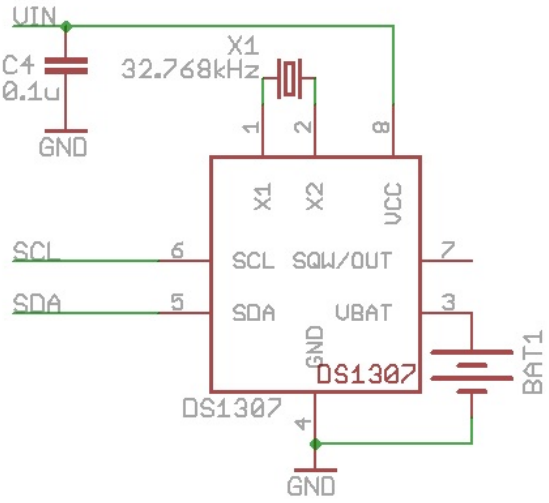


Figure 4.8: Real-time Clock

4.2.5 Serial Port

The serial port (TTL-RS232) is a very important part of the communication in the BBB, so it had to be present in the board. Therefore after some research, it was found a perfect implementation for it. The BeagleBone Black RS-232 Serial Micro-Cape, from Logic Supply, Inc. [23] had all the features needed but it had to be divided in two circuits (same schematic but different BBB connections - P9 and J) for each serial port as Figure 4.9 demonstrates.

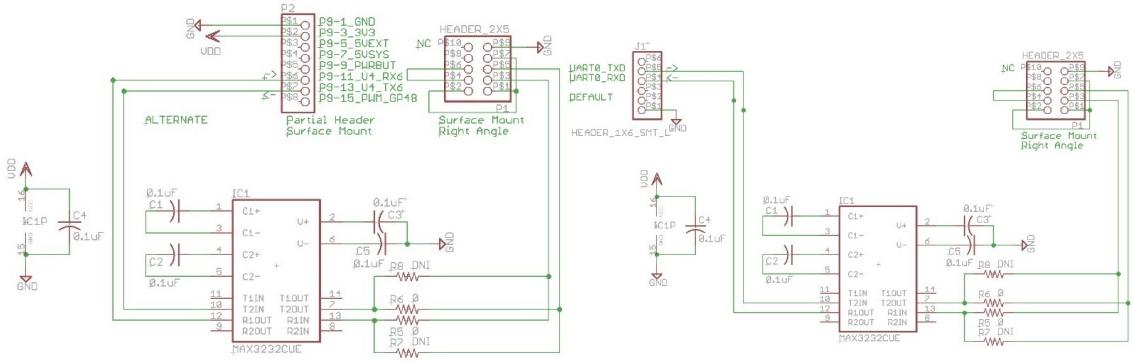


Figure 4.9: TTL-RS232 Serial Port for P9 and J1

4.2.6 DC-DC Converter

To power the BBB, if not powered by USB port, the board must have a 5V DC supply to link to the respective pin. Since the board has a 24V DC input, all that is needed is a 24V to 5V DC-DC converter. The Figure 4.10 shows a 24VAC-5VDC converter from OpenSprinkler [15], that also works with 24VDC of input.

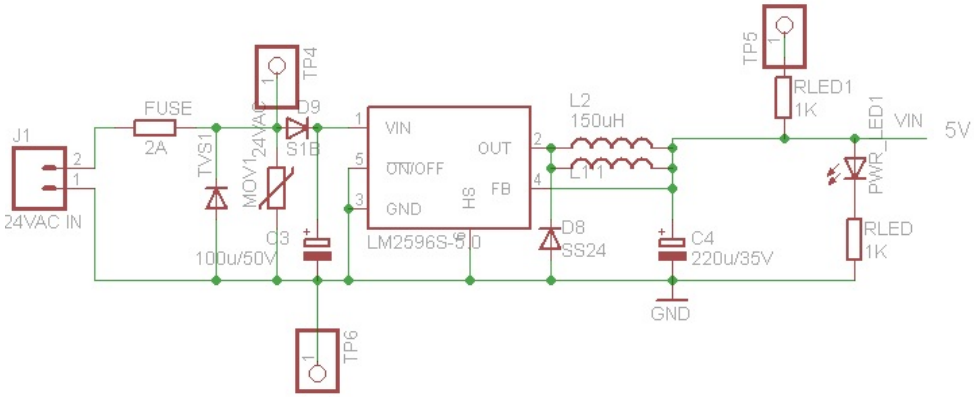


Figure 4.10: DC-DC Converter

4.3 Printed Circuit Board

After finalizing the general schematic of its components it was now time to proceed to the detailed parts of the global scheme. All the components had to be connected to were they should be connected so the result could fulfil the goals for this I/O Interface. In Figure 4.11 we can see the "silk" of the board and the complete schematic is present in B.1.

The final outlook of the PCB is shown and detailed in Figure 4.12. All the components work properly and the board is ready to be a part of the Smart Object.

Figure 4.12 Components:

1. Power Input - 24V DC
2. Outputs
3. Digital Inputs
4. Analog Inputs
5. BeagleBone Black connectors
6. Serial Ports
7. RTC Battery

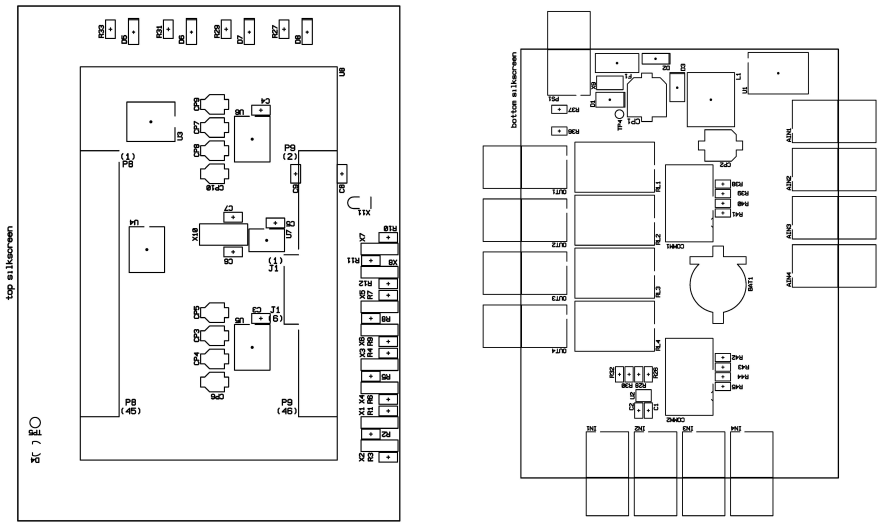


Figure 4.11: BeagleBoneBlack Top and Bottom Silk Screen

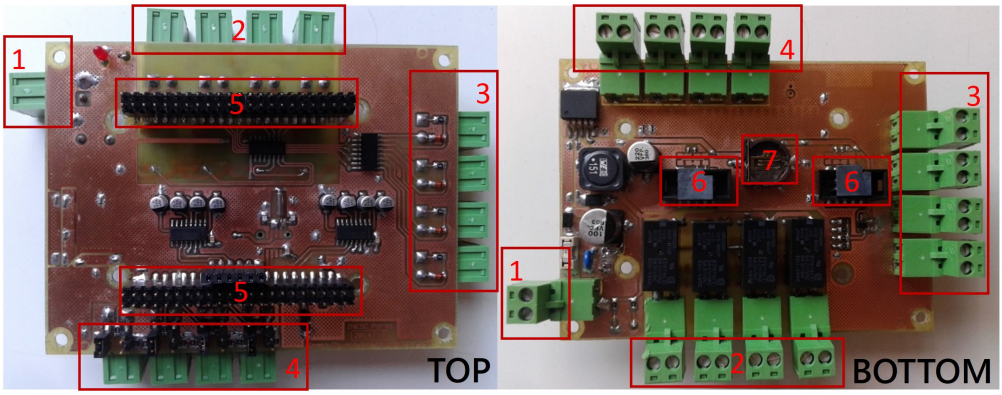


Figure 4.12: Printed Circuit Board

4.4 Conclusion

This chapter concerned the sensoring component of the Smart Object. It began with the need to have something that could physically connect the processor with the industrial environment and, since nothing in the market could suit the purpose intended, it ended being a major part of this project - the development of a complete interface for SO framework. Started from a basic concept and the more it was done, the more it was obvious the need to add other components. It is described the process of idealization and schematization of an I/O Interface that culminated in the production of a PCB with the required features such as digital and analogical inputs, relay controlled outputs and serial ports. After some tests the PCB proved to be capable of performing all the goals previously established.

Chapter 5

Communication

This chapter covers the study and implementation of different types of communications between distinct technologies and its interrelation to compose the Smart Object.

5.1 BeagleBone Black - PC

The communication between the BBB and the PC (Linux running in Virtual Machine VMware Workstation© was by USB connection, using Secure Shell (SSH) through the command line as in Figure 5.1. [24]. This process also included File Transfer Protocol (FTP), also using SSH.

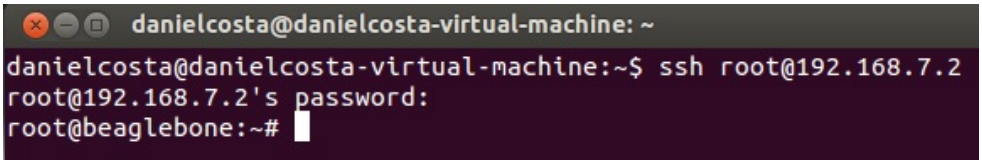


Figure 5.1: Connection with BBB through SSH

5.2 Switch Network

The main communication protocol between all components is Ethernet. In order to connect every element of the SO to build a local network is used a Ethernet Switch (SWEEX SW105[1]), that enables the addition of new and different components.

[1]More info at: http://www.sweex.com/en/assortiment/internet-networking/switches/sw105/

Figure 5.2: Actual Switch Network

5.2.1 BBB - PLC: FINS Protocol

The Factory Interface Network Service (FINS) Protocol[2] is a network protocol used by Omron PLCs (in this case the OMRON SYSMAC CP1L[3]), over different physical networks such as Ethernet, Controller Link, DeviceNet and RS-232C. The FINS communications service enables client control of operations such as reading or writing server PC memory area data without the need to program these operations into the server PC user program. The Ethernet Unit uses a dedicated UDP/IP port to execute the FINS communications service, in this case being **192.168.250.1** [25].

5.2.1.1 Program

To communicate with the Omron PLC and access its I/O data memory, a existing C++ program that read words memory in the PLC (belonging to INESC TEC Porto), was modified to access the inputs and outputs part of the PLC's data memory. That program created a executable file that when executed inside the Server (processor) it returned the data area present in the PLC memory [26].

```
1 //============================================================================
2 // Name        : omroncommunication.cpp
3 //============================================================================
4
5 #include <iostream>
6 #include <unistd.h>
7
```

[2]More details at: http://www.protocessor.com/solutions/key-protocols/Omron-FINS.php

[3]See more at: http://industrial.omron.pt/pt/products/catalogue/automation_systems/programmable_logic_controllers/compact_plc_series/cp1l/default.html

```
#include "finscommunicator.h"
using namespace std;

int main() {
        finscommunicator *onromconnection = new finscommunicator();
        //while(true)
        // {
                onromconnection->readFromPLC("192.168.250.1",9600,0,2,0xB0);
        //      usleep(1000000);
        //      cout << endl;
         //}
        return 0;
}
```

Listing 5.1: Omron Communication

The data memory address read is the CIO area -0xB0- as we can see from the code below (from finscommunicator.cpp, full code at C.1) and from the Operational Manual of OMRON SYSMAC CP1L [27].

```
int finscommunicator::memory_area_designation_code(const int addr)
{

        if(addr>0x0BFFF&&addr<0x0D800) //B0 , CIO area
                return 0xB0;
        if(addr>0x0DDFF&&addr<0xE000) //work area
                return 0xB1; //TODO: clarified code , from memory area
                    designation codes table
        if(addr>0xFFFF&&addr<0x18000) //DM area
                return 0x82;
        if(addr>0xB9FF&&addr<0xBC00) //A448 - A959
                return 0xB3;
        else
                return 0x80; //CIO,LR,HR,AR area*/
}
```

Listing 5.2: Fins Communicator Memory Area

5.3 Conclusion

For the Smart Object perspective, the communication is crucial for its proper operation. This chapter covered the connection between the BBB and the PC (via USB using SSH) and the network created for the BBB-PLC-PC using Ethernet. For the data exchange between the BBB and the PLC was used FINS Protocol for OMRON PLCs, particularly an existing program previously used for memory area reading, but that, with some alterations, is now able to read the Input/Output area of the same PLC.

Chapter 6

Processing

This chapter discusses the processing part of the Smart Object, its ”Brain”. Its is divided in four sections, being the first two the presentation of the programming language and the database used. The last two sections concern the two different Python programs used for the rule interpretation task.

6.1 Introduction

For this crucial part of the Smart Object were created two different programs: one for gathering the input information and other to interpret and assess the rules. The inputs script is responsible to read the inputs from both the PLC and the I/O Interface and, from there, count and verify them for changes and start the respective timers. When it detects a change it writes them, except the timer, in a log table in the database. The timer is inserted in a different table just for the timers and that it is actualized every time the while cycle is executed. The rules script reads the inputs, changes, counters and timers from the database and saves the results in variables that coincide with the variables present in the rules.

6.2 Python

Python is a high-level and open source programming language designed to be easy to read and simple to implement. Its design emphasizes code readability and its syntax allows express concepts in fewer lines of code than other languages such as C. TechTerms characterizes it as: [28]

> *It is open source, which means it is free to use, even for commercial applications. Python can run on Mac, Windows, and Unix systems and has also been ported to Java and .NET virtual machines.*
>
> *Python is considered a scripting language, like Ruby or Perl and is often used for creating Web applications and dynamic Web content. It is also supported by a number of 2D and 3D imaging programs, enabling users to create custom plug-ins and*

extensions with Python. Examples of applications that support a Python API include GIMP, Inkscape, Blender, and Autodesk Maya.

Scripts written in Python (.PY files) can be parsed and run immediately. They can also be saved as a compiled programs (.PYC files), which are often used as programming modules that can be referenced by other Python programs.

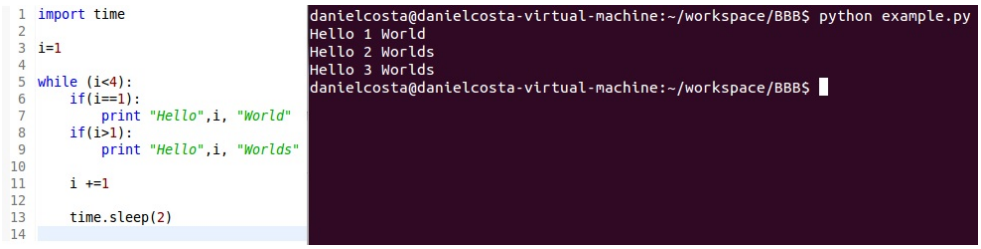


Figure 6.1: Example of Python script and result

The core philosophy of the language is summarized by the document "PEP 20 (The Zen of Python)", which includes aphorisms such as: [29]

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.

Some of its main features are described in [30], and include:

- Simple;
- Easy to Learn;
- Free and Open Source;
- Object Oriented;
- Extensive Libraries.

6.2.1 Libraries

The use of some libraries was significantly important in this project, has it allowed the simple and easy understandable access to the GPIO (Digital and Analogical) and other BBB features. The library used was the Adafruit BBIO library with several modules such as:

- `import Adafruit_BBIO.GPIO as GPIO - see example at Figure 6.2`
- `import Adafruit_BBIO.ADC as ADC`

- `import Adafruit_BBIO.UART as UART`

```
import Adafruit_BBIO.GPIO as GPIO

GPIO.setup("P8_10", GPIO.OUT)
GPIO.output("P8_10", GPIO.HIGH)
GPIO.cleanup()
```

Figure 6.2: Adafruit.GPIO Library example

6.3 Database

A SQLite3 [1] database was used in order to store the information in a log style and to interconnect the two python scripts. With the database were created the tables presented below and the file present in the same directory of the scripts. [31]

It was also used `pickle` to exchange variables between the two scripts presented in the next sections. `Pickle` is the standard mechanism for object serialization; pickling is the common term among Python programmers for serialization (unpickling for deserializing). It uses a simple stack-based virtual machine that records the instructions used to reconstruct the object. [32]

6.3.1 Rules

The rules table contains the rules and respective outputs. It his divided in three columns: id, condition and output. Minding that, we can see in Figure 6.3 some rules created for test purposes and the respective explanation:

1. **int(I_PLC[0][0])==1 and int(I_PLC[0][1])==0** → ***LED***
   This rule activates an external LED (connected to one of the relay outputs), when the PLC's I00 input is on and the I01 is off.

2. **int(I_PLC[0][1])==1** → ***email***
   An email is sent with a previous defined message when PLC's I01 is on.

3. **int(I_PLC[0][2])==1** → ***Output 1***
   The Output1 of the Interface is activated, when I02 is on.

4. **T_PLC[0][0]*int(I_PLC[0][0])>10** → ***LED***
   If I00 is on for more than ten seconds, the LED is activated.

[1] More info at `http://www.sqlite.org/index.html`

5. **C_PLC[0][0]>3,TC_PLC[0][0]<30** → ***LED***
   Whenever I00 is on a timer starts until thirty seconds pass. In that time if the I00 is on more than three times (this meaning 3 REs), the LED is turned on.

6. **D_INT[3]==1** → ***Output 2***
   The Output 2 is activated when the digital input 3 is on.

7. **A_INT[0]>3** → ***LED***
   When the voltage on the analogic input 1 exceeds 3V, the external LED is activated.

8. **C_INT[0][3]>2,TC_INT[0][3]<20** → ***email***
   Same rule as rule 5 but, in this case, it is used the digital input 3 that, when activated more than two times in 20 seconds an email is sent.

Table: rules

	id	condicao	acao
1	1	int(I_PLC[0][0])==1 and int(I_PLC[0][1])==0	LED
2	2	int(I_PLC[0][1])==1	email
3	3	int(I_PLC[0][2])==1	1
4	4	T_PLC[0][0]*int(I_PLC[0][0])>10	LED
5	5	C_PLC[0][0]>3,TC_PLC[0][0]<30	LED
6	6	D_INT[3]==1	2
7	7	A_INT[0]>3	LED
8	8	C_INT[0][3]>2,TC_INT[0][3]<20	email

Figure 6.3: Database Rules table

6.3.2 Logs

The **logs** table keeps the historic of inputs (from the PLC and Interface) and they respective changes and counters. As we can see in Figure 6.4 we have 8 columns:

- **id** - identifier of the entry;
- **input_PLC[0-3][0-7]** - array of four arrays with eight position each that assigns '1' when input detected and '0' when not. For example in line/id 408, the I[0][0] (I00) input is on.
- **changes_PLC[0-3][0-7]** - assigns '1' when there exists a rising edge (RE) in the respective input, '-1' when a falling edge (FE) happens and '0' when it stays the same as the previous

cycle. In line 408 a RE takes place and ’1’ is assigned to changes_PLC[0][0], but in the next line has the input goes to off, a ’-1’ appears in the same position;

- **counter_PLC[0-3][0-7]** - increments the REs of the inputs, but only if a rule using counter of that input is detected. It resets when the timer of counter exceeds the time interval defined in the rule. An example is present from line 404 to 410 where the input I00 is activated 3 times, appearing the ’3’ in counter_PLC[0][0];

- **input_INT[0-1][0-3]** - two arrays of four positions each one containing the digital inputs and other the analogic. The digital inputs (input_INT[0][0-3]) is works just like the input_PLC, ’1’ is input on and ’0’ is off. In line 410 a ’1’ is at the position [0][3], that meaning that the digital input 3 is activated. The analog inputs (input_INT[1][0-3]) goes from ’0’ to ’10’ as it represents the voltage on the respective input (0V-10V). From line 400 to 404 when can see the ADC1(input_INT[1][0]) working;

- **changes_INT[0-1][0-3]** - as changes_PLC it illustrates the changes that occur in the inputs. In changes_INT[0][0-3] (digital inputs) assigns ’1’ for RE, ’-1’ for FE ad ’0’ if nothing happens. As for the analogical inputs, changes_INT[1][0-3], places a ’1’ when the voltage increases, ’-1’ when it decreases and ’0’ when it stays the same as previous cycle. In the figure we can see from line 400 to 404 that the ADC1 is on and, from there, when the voltage increases, a ’1’ appears in change_INT[1][0] and when the voltage drops a ’-1’ is in that same position;

- **counter_INT[0-1][0-3]** - exact same concept as the counter_PLC but for the Interface inputs. Example at line 410, when input_INT[0][3]=1, counter_INT[0-1][0-3]=0+1.

- **date** - date and time. In Figure 6.4 the date is only actualized on line 405;

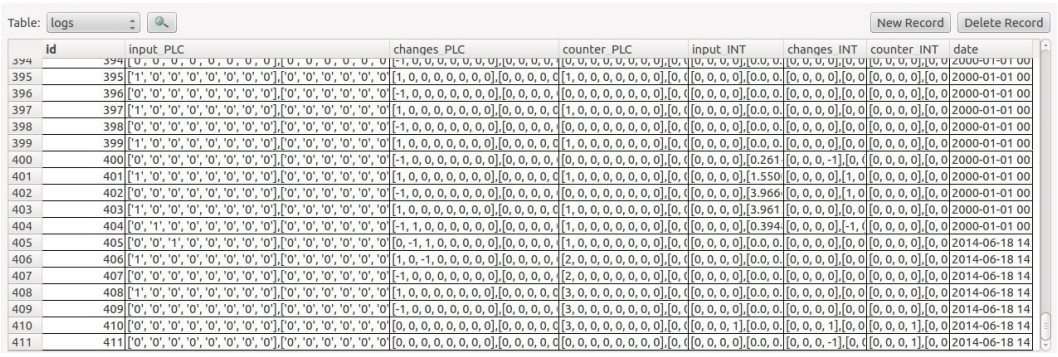


Figure 6.4: Database Logs table

6.3.3 Timers

In order to save the timers information without risking the loss or overlapping it was decided to store this information into an exclusive table. This table has 5 columns, as shown in Figure 6.5:

- **id** - identifier;
- **timer_PLC** - timer that starts when it is detected a RE on an input and stops and resets when that a FE is detected in that same input. In other words, the timer runs only when an input is on.
- **timer_counter_PLC** - timer relative with the counter condition (for example rule number 5 or 8), that starts with a RE and lasts until the seconds in the condition had past. For example, in rule number 5 it starts with a RE on I00 and lasts 30 seconds, then it resets.
- **timer_INT** - same operating logic as timer_PLC but for the inputs on the I/O Interface.
- **timer_counter_PLC** - also same working condition as timer_counter_PLC but for I/O Interface inputs.

Note that all timers only start if a condition with that timer exists.

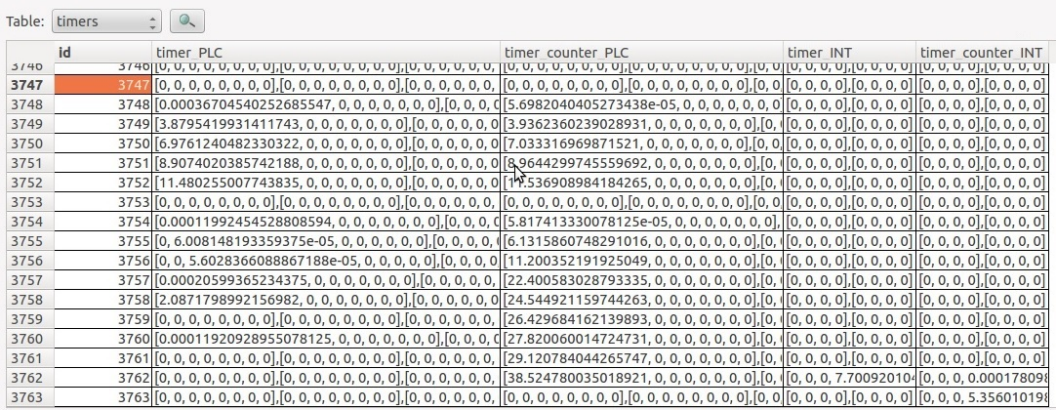


Figure 6.5: Database Timers table

Having now the database ready it is time to present and demonstrate the two scripts responsible for the processing part of the SO.

6.4 Inputs

This script is responsible for gathering the information from the ”outside” of the Smart Object, this being the surrounding environment to it. Analysing the script(fully available at C.2.1) we have:

- Variable and library declaration;
- Core processing part - `main()` - which can also be divided in (readily apparent in the code):
  - PLC
  - Interface

Since the first point is a complement for the core functions of the script and can be easily understood it is not going to be a target of a detailed explanation. Minding that, it is only important for the rest of code to define the BBB pins of the inputs, being that a simple script, thanks to Adafruit GPIO library:

```
DIN1="P8_29"
DIN2="P8_31"
DIN3="P8_33"
DIN4="P8_35"
AIN1='P9_40'
AIN2='P9_37'
AIN3='P9_38'
AIN4='P9_33'
#-------------PIN DECALRATION--------------
GPIO.cleanup()
GPIO.setup("P8_29", GPIO.IN)
GPIO.setup("P8_31", GPIO.IN)
GPIO.setup("P8_33", GPIO.IN)
GPIO.setup("P8_35", GPIO.IN)
ADC.setup()
```

The second point starts in `main()` function that calls the function `get()` of class `GetData`.

```
def main():
    plc=GetData()
    while(1):
        plc.get()
        time.sleep(0.1)
        #raw_input("Press Enter to continue...")
```

The class `GetData` is divided in two functions: `db()` and `get()`. through out the code in PLC and Interface, as we can see from the segments of code displayed below:

- **get function**
  - *Get data from PLC*

    Runs the executable presented before in section 5.2.1.1 and returns the PLC data, passed as the argument of `db` function.

```
def get(self):
            args = ("./omron")
            popen = subprocess.Popen(args, stdout=subprocess.
                PIPE)
            popen.wait()
            output = popen.stdout.read()
            a=eval(output)

            self.db(a)
```

- **db function**

– *Pickle load*

Defines a new function `db`. Loads the variables flag_PLC, flag_INT, flag_T_PLC, flag_T_INT, flag_C_PLC and flag_C_INT from `shared.pkl` file, from rules.py script.

```
        def db(self, number):
            with open("shared.pkl",'rb') as fp:
                flag_PLC, flag_INT, flag_T_PLC, flag_T_INT,
                    flag_C_PLC, flag_C_INT = pickle.load(fp)
            fp.close()
```

– *Database connection*

Using Sqlite3, connects to rules.db and a cursor. That cursor is then used to fetch the id number of the last log and increment one to have the id for a new line.

```
            conn = sqlite3.connect("rules.db",timeout=10)
            cursor = conn.cursor()
            cursor.execute("SELECT id FROM logs")
            result=cursor.fetchall(
            leng=len(result)
            input_id=leng+1
```

– *Inputs PLC*

Receives the data from `get()` function, converts it to int, append them to the I vector for after conversion to string (to input into the database).

```
            I=[]
            j=0
            while (j<len(number.keys())):
                i=0
                a=int(number['I%d' % j])
                b=format(a, '010b')
                c=b[2:]
                I.append([])

                while(i<len(c)):
                    I[j].append(c[(len(c)-1)-i])
                    i=i+1
                j=j+1

            str_PLC = ','.join(str(e) for e in I)  #convert
                array of inputs to string
```

– *Inputs Interface*

Digital Inputs

The raw inputs are '1' if nothing detected and '0' if activated. Then, the script, reads the inputs from 29 to 35 and, if '0' detected a '1' is assigned to the respective position on the `D_INT` vector, that, when finished, is appended to the `I_INT` vector.

```
            I_INT=[]
            h=1
            DIN=[]
```

```
            d=29
            while(h<5):
                f=GPIO.input("P8_%d" % d)
                if(f==0):
                    value=1
                elif(f==1):
                    value=0
                DIN.append(value)
                d=d+2
                h=h+1
            I_INT.append(DIN)
```

Analog Inputs

This part of the script reads the value of the ADC (from 0 to 1) and then multiples it by 10, to make the 0-10V input. In the end it appends the `A_INT` vector to the `I_INT` and the converts it to string.

```
            h=1
            AIN=[]
            while(h<5):
                ADC.read('AIN%d' % h)
                value = ADC.read('AIN%d' % h)
                voltage = value * 10 #10V
                AIN.append(voltage)
                h=h+1
            I_INT.append(AIN) #vector com os inputs da
                interface

            str_INT = ','.join(str(e) for e in I_INT)
```

– *Changes PLC*

First, the previous value of the input must be known: if the table is empty the A vector is filled with zeros; if not, the column `input_PLC` from the last row from logs is fetched. This vector is then compared with the previous `I_PLC` vector to verify changes. If a RE is detected a '1' is assigned, if a FE detected a '-1' is assigned and is the value is equal to previous value a '0' is appended to the `J` vector, then converted to string at the end.

```
        if(leng==0):
            A=[[0 for col in range(8)] for row in range(4)]
        else:
            cursor.execute("SELECT input_PLC FROM logs WHERE
                id=?",(leng,))
            x=cursor.fetchone()
            y=''.join(x)
            A=eval(y)
        n=len(I)
        f=0
        J=[]
        changes_PLC=0
        while(f<n):
            b=0
            J.append([])
```

```
                while(b<8):
                    if (int(A[f][b])<int(I[f][b])):
                        J[f].append(1)
                        changes_PLC=1
                    elif (int(A[f][b])>int(I[f][b])):
                        J[f].append(-1)
                        changes_PLC=1
                    else:
                        J[f].append(0)
                    b=b+1
                f=f+1

            str_changes_PLC = ','.join(str(e) for e in J)   #
                convert array of changes to string
```

– *Changes INT*

The code is similar to Changes PLC in 6.4, but this one prevents the inexistence of an Input_INT in the last fetched row. It also compares the integer of the ADC and assigns the '1','-1' or '0' according to, respectively, increase, decrease or unaltered ADC value.

```
            if(leng==0):
                B=[[0 for col in range(4)] for row in range(2)]
            else:
                cursor.execute("SELECT input_INT FROM logs WHERE
                    id=?",(leng,))
                x=cursor.fetchone()
                if(x[0]==None):
                    B=[[0 for col in range(4)] for row in range
                        (2)]
                else:
                    y=''.join(x)
                    B=eval(y)
            f=0
            J_INT=[]
            changes_INT=0
            while(f<2):
                b=0
                J_INT.append([])
                while(b<4):
                    if (int(B[f][b])<int(I_INT[f][b])):
                        J_INT[f].append(1)
                        changes_INT=1
                    elif (int(B[f][b])>int(I_INT[f][b])):
                        J_INT[f].append(-1)
                        changes_INT=1
                    else:
                        J_INT[f].append(0)
                    b=b+1
                f=f+1

            str_changes_INT= ','.join(str(e) for e in J_INT)   #
                convert array of changes to string
```

– *Counter PLC*

The first part of the code increments one to the vector `G` position where the RE happens. Then, and if a Timer Counter condition is detected in rules (`flag_T_PLC`) three situations may occur: if a RE is detected and the start time is zero, the start time is actualized and the timer will start; if the start time is zero and nothing happens to that input, it will remain zero; for last if neither of the previous situations happen the timer counter value is actualized by subtracting the start time to the current time. When the `flag_PLC` of an input is raised, this meaning, when the time counter exceeds the time in the condition, the respective start time vector position is reset and so are the strings for the next input on the database.

```
1             f=0
2             while(f<n):
3                 b=0
4                 while(b<8):
5                     if(flag_C_PLC[f][b]):
6                         if(J[f][b]==1):
7                             G[f][b]=G[f][b]+1
8                     b=b+1
9                 f=f+1
10            print 'Input Counter'
11            print G
12            str_counter_PLC = ','.join(str(e) for e in G)
13            f=0
14            while(f<n):
15                b=0
16                while(b<8):
17                    if(flag_T_PLC[f][b]):
18                        if(start_time2[f][b]==0 and J[f][b]==1):
19                            start_time2[f][b] = time.time()
20                        if(start_time2[f][b]==0):
21                            T_counter[f][b]=0
22                        else:
23                            T_counter[f][b]=(time.time() -
                                start_time2[f][b])
24                    b=b+1
25                f=f+1
26            print 'Timer Counter Inputs'
27            print T_counter
28            timestr2 = ','.join(str(e) for e in T_counter)
29
30            changes_counter_PLC=0
31            coun=''
32            counT=''
33            f=0
34            while(f<n):
35                b=0
36                while(b<8):
37                    if(flag_PLC[f][b]==1):
38                        changes_counter_PLC=1
```

```
                        cursor.execute("SELECT counter_PLC FROM
                            logs WHERE id=(SELECT MAX(id) FROM
                            logs)")
                        x=cursor.fetchone()
                        y=''.join(x)
                        coun=eval(y)
                        coun[f][b]=0
                        G[f][b]=0
                        cursor.execute("SELECT timer_counter_PLC
                             FROM timers WHERE id=(SELECT MAX(id
                            ) FROM timers)")
                        x=cursor.fetchone()
                        y=''.join(x)
                        counT=eval(y)
                        counT[f][b]=0
                        start_time2[f][b]=0
                        flag_PLC[f][b]=0
                    b=b+1
                f=f+1
            if(changes_counter_PLC):
                str_counter_PLC=','.join(str(e) for e in coun)
                timestr2=','.join(str(e) for e in counT)
```

– *Counter INT*

Exact same operation principle as the Counter PLC present in 6.4, but with different variable and flag names.

```
            f=0
            while(f<2):
                b=0
                while(b<4):
                    if(flag_C_INT[f][b]):
                        if(J_INT[f][b]==1):
                            G_INT[f][b]=G_INT[f][b]+1
                    b=b+1
                f=f+1
#             print 'Input Counter INTERFACE'
#             print G_INT
            str_counter_INT = ','.join(str(e) for e in G_INT)

            f=0
            while(f<2):
                b=0
                while(b<4):
                    if(flag_T_INT[f][b]):
                        if(start_time2_INT[f][b]==0 and G_INT[f
                            ][b]==1):
                            start_time2_INT[f][b] = time.time()
                        if(start_time2_INT[f][b]==0):
                            T_counter_INT[f][b]=0
                        else:
                            T_counter_INT[f][b]=(time.time() -
                                start_time2_INT[f][b])
                    b=b+1
                f=f+1
```

```
#               print 'Timer Counter Inputs INTERFACE'
#               print T_counter_INT
              timestr2_INT = ','.join(str(e) for e in
                  T_counter_INT)

              changes_counter_INT=0
              coun=''
              counT=''
              #print flag_INT
              f=0
              while(f<2):
                  b=0
                  while(b<4):
                      if(flag_INT[f][b]==1):
                          changes_counter_INT=1
                          cursor.execute("SELECT counter_INT FROM
                              logs WHERE id=(SELECT MAX(id) FROM
                              logs)")
                          x=cursor.fetchone()
                          y=''.join(x)
                          coun_INT=eval(y)
                          coun_INT[f][b]=0
                          G_INT[f][b]=0
                          cursor.execute("SELECT timer_counter_INT
                               FROM timers WHERE id=(SELECT MAX(id
                              ) FROM timers)")
                          x=cursor.fetchone()
                          y=''.join(x)
                          counT_INT=eval(y)
                          counT_INT[f][b]=0
                          start_time2_INT[f][b]=0
                          flag_INT[f][b]=0
                      b=b+1
                  f=f+1
              if(changes_counter_INT):
                  str_counter_INT=','.join(str(e) for e in
                      coun_INT)
                  timestr2_INT=','.join(str(e) for e in counT_INT)
```

– *Write on Logs table*

Detects if there where changes on the inputs or timers of both PLC and Interface and insert them into the logs table.

```
              #ESCREVER NA BASE DE DADOS#
              datime=time.strftime('%Y-%m-%d %H:%M:%S')
              if(changes_PLC or changes_INT or changes_counter_PLC
                   or changes_counter_INT):
                  cursor.execute("INSERT INTO logs VALUES
                      (?,?,?,?,?,?,?,?)",(input_id,str_PLC,
                      str_changes_PLC,str_counter_PLC,str_INT,
                      str_changes_INT,str_counter_INT,datime))
                  conn.commit()
```

– *Timer PLC*

This part of the script saves the start time when a RE and the Input is detected and reset when a FE happens. Meanwhile, and with the input still on, the respective position in vector `T` is actualized in every cycle with the difference between the current time and the start time.

```
1                f=0
2                while(f<n):
3                    b=0
4                    while(b<8):
5                        if(int(J[f][b])==1 and int(I[f][b])==1):
6                            start_time[f][b] = time.time()
7                        elif(int(J[f][b])==-1):
8                            start_time[f][b]=0
9
10                       if(int(I[f][b])==1):
11                           T[f][b]=(time.time() - start_time[f][b])
12
13                       else:
14                           T[f][b]=0
15                       b=b+1
16                   f=f+1
17               print 'Timer Inputs'
18               print T
19               timestr= ','.join(str(e) for e in T)  #convert array
                      of timers to string
```

– *Timer INT*

Same logic operation as the Timer PLC, with different variables (interface variables).

```
1                f=0
2                while(f<2):
3                    b=0
4                    while(b<4):
5                        if(int(J_INT[f][b])==1 and int(I_INT[f][b])
                             ==1):
6                            start_time_INT[f][b] = time.time()
7                        elif(int(J_INT[f][b])==-1):
8                            start_time_INT[f][b]=0
9
10                       if(int(I_INT[f][b])==1):
11                           T_INT[f][b]=(time.time() -
                                 start_time_INT[f][b])
12
13                       else:
14                           T_INT[f][b]=0
15                       b=b+1
16                   f=f+1
17   #             print 'Timer Inputs INTERFACE'
18   #             print T_INT
19               timestr_INT= ','.join(str(e) for e in T_INT)   #
                     convert array of timers to string
```

– *Write on Timers table*
After the timer actualization the id of the timer is incremented and a new row is inserted into the timers table.

```
1            cursor.execute("SELECT id FROM timers")      #ID do
                 timer
2            result=cursor.fetchall()                     #ID do
                 timer
3            leng=len(result)                             #ID do
                 timer
4            timer_id=leng+1                              #ID do
                 timer
5
6            cursor.execute("INSERT INTO timers VALUES
                 (?,?,?,?,?)",(timer_id,timestr,timestr2,
                 timestr_INT,timestr2_INT))
7            conn.commit()
```

6.5 Rules

The purpose of this python script is to read and interpret the rules in the database and to activate the outputs. For that it is divided in those two parts: rule interpretation and output activation. Analysing the code, we first have the pin configuration and variable declaration:

```
1  'OUTPUT1','P8_12'
2  'OUTPUT2','P8_14'
3  'OUTPUT3','P8_16'
4  'OUTPUT4','P8_18'
5  #------------------------------------------------------------------#
6  global flag_PLC
7  global flag_INT
8  global flag_T_PLC
9  global flag_T_INT
10 global flag_C_PLC
11 global flag_C_INT
12 flag_PLC=[[0 for col in range(8)] for row in range(4)]
13 flag_T_PLC=[[0 for col in range(8)] for row in range(4)]
14 flag_C_PLC=[[0 for col in range(8)] for row in range(4)]
15 flag_INT=[[0 for col in range(4)] for row in range(2)]
16 flag_T_INT=[[0 for col in range(4)] for row in range(2)]
17 flag_C_INT=[[0 for col in range(4)] for row in range(2)]
18 #------------------DEFINIR PINOS
       ----------------------------------------#
19 GPIO.setup('P8_12', GPIO.OUT)
20 GPIO.setup('P8_14', GPIO.OUT)
21 GPIO.setup('P8_16', GPIO.OUT)
22 GPIO.setup('P8_18', GPIO.OUT)
```

Apart from this and the main, where a infinite cycle is running and the flags are reset each end of a cycle, we have the class `GetRules` divided in two functions:

- **rules function**

– *Inputs*)

In this part of the script, apart from the database connection, the inputs of both PLC and Interface are fetched from the database and assigned to the respective vectors.

```
conn = sqlite3.connect("rules.db", timeout=10)
cursor = conn.cursor()
cursor.execute("SELECT input_PLC FROM logs WHERE id
    =(SELECT MAX(id) FROM logs)")
x=cursor.fetchone()
y=''.join(x)
I_PLC=eval(y)#I_PLC
cursor.execute("SELECT input_INT FROM logs WHERE id
    =(SELECT MAX(id) FROM logs)")
x=cursor.fetchone()
y=''.join(x)
I_INT=eval(y)#I_INT
D_INT=I_INT[0]#Digital Inputs
A_INT=I_INT[1]#Analog Inputs
```

– *Rule fetch*

Through this code it is possible to place all rules in one array(`result`), and assess different rules by the array position.

```
cursor.execute("SELECT * FROM rules")
result=cursor.fetchall()
leng=len(result)
i=0
```

– *Condition analysis*

This while cycle analyses every position in the `result` array, creates the variables needed to evaluate the condition and the activates flags if necessary. A `special` flag is activated if the rule is a counter rule.

```
while(i<leng):
    res=result[i]
    idd=res[0]
    condicao=res[1]
    s_n=0
    s_counter=[0,0]
    s_counter_INT=[0,0]
    special=0
```

Counter

If a counter rule is detected (PLC in the first case and Interface in the second) this code will select the respective counter and timer counter and save them in arrays. This rule activates the `special` flag.

```
if 'C_PLC' in condicao:#C_PLC
    special=1
    cursor.execute("SELECT counter_PLC FROM
        logs WHERE id=(SELECT MAX(id) FROM
        logs)")
    x=cursor.fetchone()
```

```
5                       w=''.join(x)
6                       C_PLC=eval(w)#C_PLC
7
8                       cursor.execute("SELECT timer_counter_PLC
                            FROM timers WHERE id=(SELECT MAX(id)
                            FROM timers)")
9                       x=cursor.fetchone()
10                      w=''.join(x)
11                      TC_PLC=eval(w)#TC_PLC
12
13
14                  if 'C_INT' in condicao:#C_INT
15                      special=1
16                      cursor.execute("SELECT counter_INT FROM
                            logs WHERE id=(SELECT MAX(id) FROM
                            logs)")
17                      x=cursor.fetchone()
18                      w=''.join(x)
19                      C_INT=eval(w)#C_INT
20
21                      cursor.execute("SELECT timer_counter_INT
                            FROM timers WHERE id=(SELECT MAX(id)
                            FROM timers)")
22                      x=cursor.fetchone()
23                      w=''.join(x)
24                      TC_INT=eval(w)#TC_INT
```

Timer

When a timer rule is detected, the PLC or Interface timer is selected from the timers table. This is not a special rule, so a `eval` is enough to evaluate it, as we can see in 6.5.

```
1                   if 'T_PLC' in condicao:#T_PLC
2                       cursor.execute("SELECT timer_PLC FROM
                            timers WHERE id=(SELECT MAX(id) FROM
                            timers)")
3                       x=cursor.fetchone()
4                       w=''.join(x)
5                       T_PLC=eval(w)#T_PLC
6
7                   if 'T_INT' in condicao:#T_INT
8                       cursor.execute("SELECT timer_INT FROM
                            timers WHERE id=(SELECT MAX(id) FROM
                            timers)")
9                       x=cursor.fetchone()
10                      w=''.join(x)
11                      T_INT=eval(w)#T_INT
```

Flags

The first and second if condition of the following script signalizes if the rule is a counter rule(`special` flag) and it assigns a '1' to a flag array(`flag_T_PLC` or `flag_T_INT`) to enable the timer counter in the inputs.py script. The last two ifs assign '1' if a counter is detected in the rule, tests the veracity of the rule with a

`eval` assigning it to an array (`s_counter[counter,timer_counter]`) and checks the condition every cycle for timeouts, in other words, this part of the script is responsible for, if the time exceeds the time setted in the rule, activate the respective flag in order reset the timer_counter in the inputs.py script.

```
                if 'TC_PLC' in condicao:
                    special=1
                    y=condicao.split(',')
                    z=re.findall(r'\d+', y[0])
                    e=int(z[0])
                    d=int(z[1])
                    flag_T_PLC[e][d]=1

                if 'TC_INT' in condicao:
                    special=1
                    y=condicao.split(',')
                    z=re.findall(r'\d+', y[0])
                    e=int(z[0])
                    d=int(z[1])
                    flag_T_INT[e][d]=1

                if 'C_PLC' in condicao:#C_PLC
                    s_counter=eval(condicao)
                    y=condicao.split(',')
                    z=re.findall(r'\d+', y[1])
                    g=int(z[0])
                    h=int(z[1])
                    flag_C_PLC[g][h]=1
                    #print s_counter
                    if (not (s_counter[1])):
                        #print 'flag'
                        flag_PLC[g][h]=1
                    else:
                        flag_PLC[g][h]=0

                if 'C_INT' in condicao:#C_INT
                    s_counter_INT=eval(condicao)
                    y=condicao.split(',')
                    z=re.findall(r'\d+', y[1])
                    g=int(z[0])
                    h=int(z[1])
                    flag_C_INT[g][h]=1
                    #print s_counter_INT
                    if (not (s_counter_INT[1])):
                        #print 'flag int'
                        flag_INT[g][h]=1
                    else:
                        flag_INT[g][h]=0
```

Regular Condition

If the rule is a non-counter rule, is considered "regular", sufficing for this an `eval` of the condition to test is veracity.

```
                if(not special):
```

```
2                       s_n=eval(condicao) #evaluates the
                            condition and returns true if equal
                            and false if different
```

– *Verify condition*

The condition verification is made through the `evals` present in the code. If a regular rule is detected an eval that evaluates the condition assigned to a simple variable (`s_n`) is enough. Yet, if a counter condition is detected the eval is stored into the s_counter/s_counter_INT, being the rule truth if the first (counter) and the second position (timer counter not exceeding the time limit setted) are both True. After that, all the ”condition truth flags” are evaluated in the if condition and if correct a ’1’ is assigned to the first position of the `acao` array (`acao[1]`) and the respective action is always stored in second position (`acao[1]`) of that same array. In the end, the action function is called, having the `acao` array as the argument.

```
1                   acao=[0,0]
2                   if(s_n or (s_counter[0] and s_counter[1]) or (
                        s_counter_INT[0] and s_counter_INT[1])):
3                       acao[0]=1
4                       acao[1]=res[2]
5                       print 'Condicao %d correcta!' % idd #
                            Condition CHECK!
6                       self.action(acao)
7                   else:
8                       acao[0]=0
9                       acao[1]=res[2]
10                      print 'Condicao %d nao correcta!' % idd #
                            Condition FAILURE!
11                      self.action(acao)
```

- **action function**

This function receives as an argument the array containing the veracity of the rule and its respective action.

– *Rule is correct*

LED ON

If the second position of `acao` is ’LED’ (string), then the LED is activated. It is necessary to specify wherein output the LED is connected, in this case being P8_18.

```
1                   if(acao[1]=='LED'):
2                       GPIO.output(('P8_18'),GPIO.HIGH)
3                       print 'LED ON'
```

Send Email

An ’email’ string must be detected in `acao[1]` to send the email. Using this code and having an Internet connection to connect to the server, an email with a programmable default message is sent from and to a desired address. [33]

```
1                   if(acao[1]=='email'):
2                       print 'Enviar Email!'
3                       import smtplib
4
5                       fromaddr = 'ee09173fe.up.pt'toaddrs =
                            'ee09173fe.up.pt'
6
7                       msg = 'There was a terrible error that
                            occured and I wanted you to know!'
8
9                       # Credentials (if needed)
10                      username = 'ee09173'
11                      password = '**********'
12
13                      # The actual mail send
14                      server = smtplib.SMTP('smtp.fe.up.pt:25'
                            )
15                      server.starttls()
16                      server.login(username,password)
17                      server.sendmail(fromaddr, toaddrs, msg)
18                      server.quit()
```

Output ON

If an integer between the range of 1 to 4 (Interface Outputs) is detected, the respective output is activated. In this example the P8_18 is deactivated as the LED is in that position.

```
1                   if (acao[1]=='1' or acao[1]=='2' or acao
                        [1]=='3' or acao[1]=='4'):
2                       output=int(acao[1])
3                       print 'Output %d ON' % output
4
5                       if(acao[1]=='1'):
6                           GPIO.output(('P8_12'),GPIO.HIGH)
7                       elif(acao[1]=='2'):
8                           GPIO.output(('P8_14'),GPIO.HIGH)
9                       elif(acao[1]=='3'):
10                          GPIO.output(('P8_16'),GPIO.HIGH)
11                      #elif(acao[1]=='4'):
12                          #GPIO.output(('P8_18'),GPIO.HIGH)
```

– *Rule is not correct*

LED OFF

Turn off the LED if condition is not correct and if action is 'LED'.

```
1                   if(acao[1]=='LED'):
2                       GPIO.output(('P8_18),GPIO.LOW)
3                       print 'LED OFF'
```

Output OFF

Power off the respective output if the condition is not correct and the action is 1 to 4 integer.

```
1                elif (acao[1]=='1' or acao[1]=='2' or acao
                     [1]=='3' or acao[1]=='4'):
2                    output=int(acao[1])
3                    print 'Output %d OFF' % output
4
5                    if(acao[1]=='1'):
6                        GPIO.output(('P8_12'),GPIO.LOW)
7                    elif(acao[1]=='2'):
8                        GPIO.output(('P8_14'),GPIO.LOW)
9                    elif(acao[1]=='3'):
10                       GPIO.output(('P8_16'),GPIO.LOW)
11                   #elif(acao[1]=='4'):
12                       #GPIO.output(('P8_18'),GPIO.LOW)
```

6.6 Conclusion

The thinking machine of the Smart Object. It is where everything is controlled, including the other components. Using Python language to program the scripts that interpret the rules and SQLite3 to set the database, the micro-processor (BBB) is now able to perform what was planned in the beginning of this project. Meaning this that, using two scripts (one to read the inputs and the other to interpret the conditions and its actions), this Smart Object for Manufacturing is ready to be put to some tests to prove its efficacy.

The Database is composed by three tables: rules, logs and timers. The rules table is where the rules are stored and divided in three columns: id, the condition that is meant to be interpret and the consequent action to perform if the condition is true. The logs table keeps the historic of inputs, changes and counters from the PLC and the I/O Interface distributed by the following columns: `id`, `inputs_PLC`, `changes_PLC`, `counter_PLC`, `inputs_INT`, `changes_INT`, `counter_INT` and `date`. Timers table is composed by the timers of the inputs and the timers of the counters, in other words, by the timers that start with a rising edge and stop/reset with a falling edge (`timer_PLC` and `timer_INT`) and by the timers for the counter rules (`inputs_counter_PLC` and `inputs_counter_INT`).

Both scripts run in a infinite while cycle and can each be divided in several parts. The `inputs.py` has the variable and library declaration and the main part, responsible for collecting and processing the PLC and Interface inputs, such as the changes in the inputs, counters and related timers. The script gathers the inputs. The `rules.py` fetches the condition and actions from the rules table, starts a while to process each rule and then, all variables needed for each condition are obtained from the other two tables of the database. For instance, if the condition is a counter condition (if a counter is present) it will fetch only the variables related to the counters (`counter_PLC/INT` and `timer_counter_PLC/INT`) from the last record of the respective table.

Chapter 7

Results

This chapter debates the results of the several tests conducted to exemplify the behaviour of the Smart Object. This results were produced by tests that were conducted by running both scripts simultaneously and manually activating the inputs of the PLC and the Interface.

7.1 Tests

As the main purpose of this project is having a functional and fully operational SO, several final tests had to be made. Using the default rules present in the database, the inputs were activated through wires from the DC converter of the Omron PLC (24V) and a voltage divider using a potentiometer to have the Analogical Input voltage range (0-10V). The results of those tests are exposed in the enumerated list bellow.

1. **int(I_PLC[0][0])==1 and int(I_PLC[0][1])==0** $\rightarrow$ ***LED***
   For this first rule, we only need to power the I00 input on the PLC to activate the LED. As we can see in the bottom console, the I_PLC[0][0] is '1', meaning that the I00 is on.

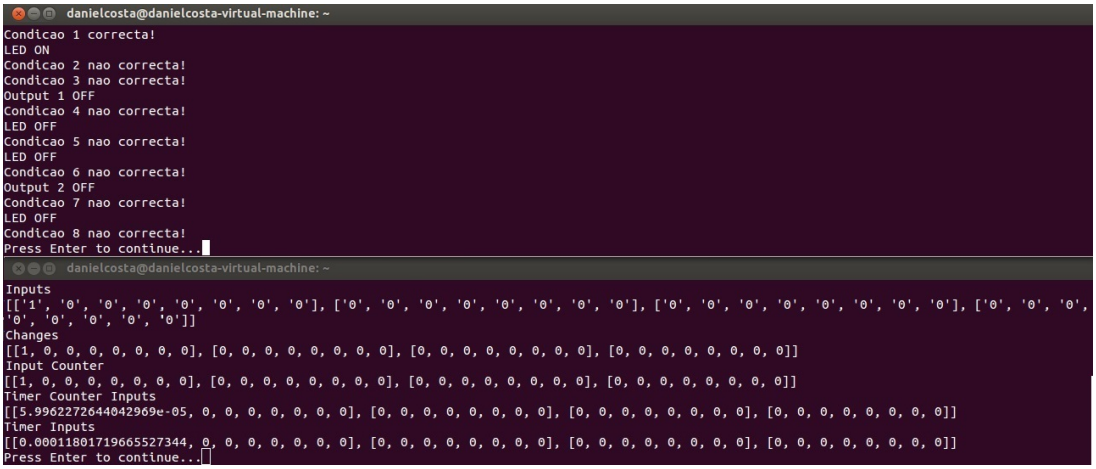


Figure 7.1: Result Condition 1

2. **int(I_PLC[0][1])==1** → ***email***

   In this case the procedure as just connecting the I01 to 24V to send the email.

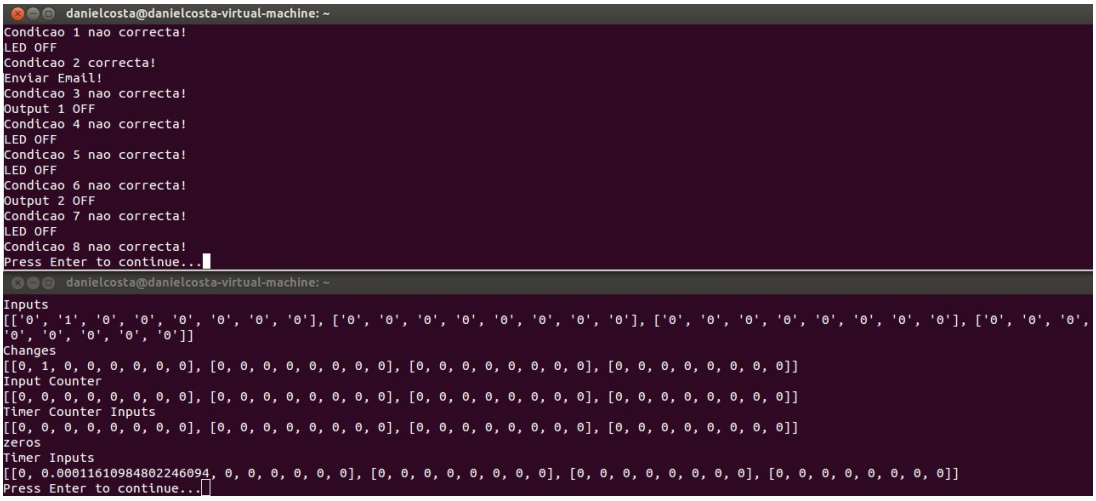


Figure 7.2: Result Condition 2

3. **int(I_PLC[0][2])==1** → ***Output 1***

   This rule activates the Output3 by powering the I02 inputs of the PLC.

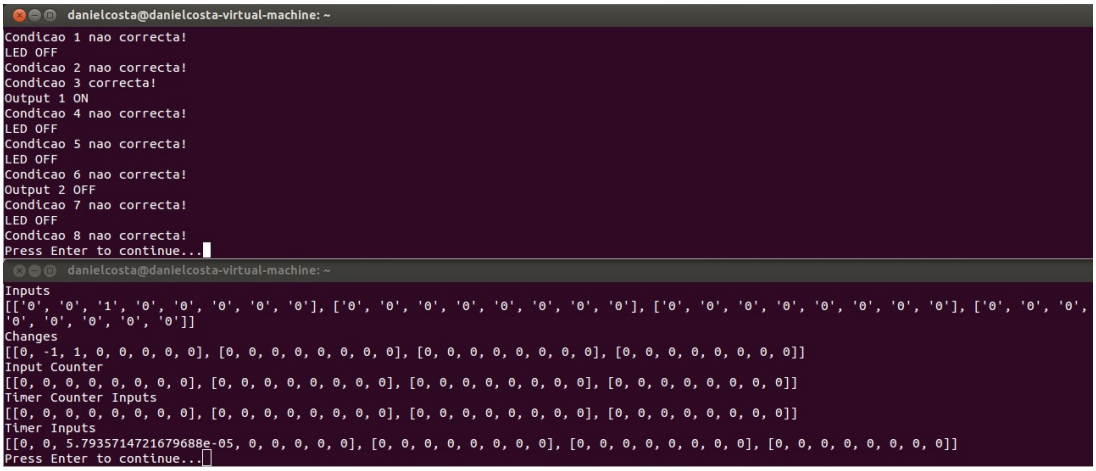


Figure 7.3: Result Condition 3

4. **T_PLC[0][0]*int(I_PLC[0][0])>10 → *LED***
   This rule is a timer rule. It activates the LED if the I00 is connected for more than 10 seconds.

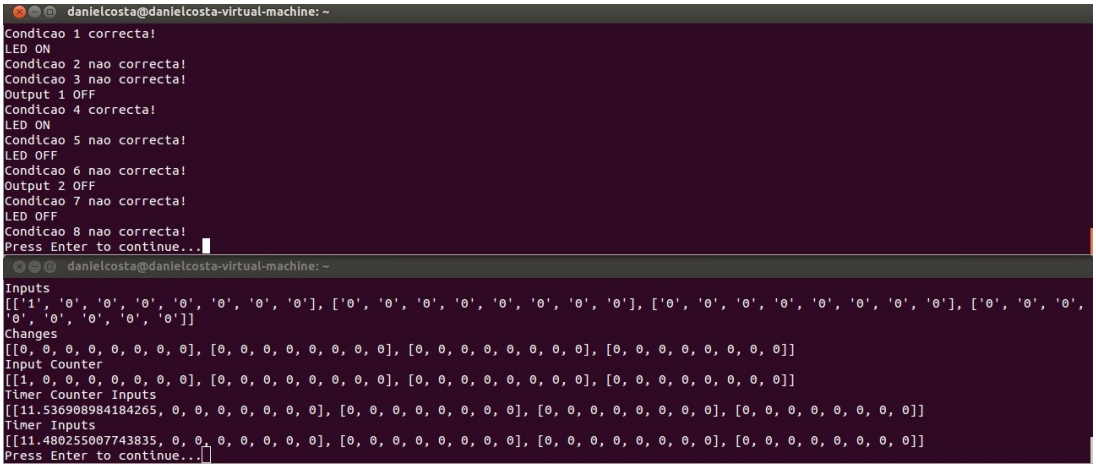


Figure 7.4: Result Condition 4

5. **C_PLC[0][0]>3,TC_PLC[0][0]<30 → *LED***
   This rule is a counter rule, if more than 3 Rising Edges in I00, in less than 30 seconds the LED is powered on. As we can see in Figure 7.5, bottom console the Input Counter of the I00 is 4, and its Timer Counter is approximately 8 seconds, so the condition is True and the LED is activated.

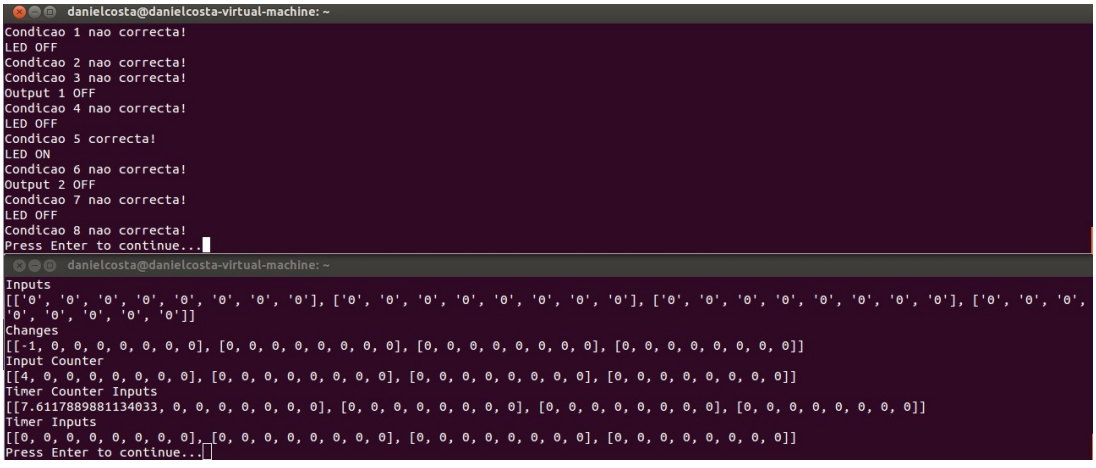


Figure 7.5: Result Condition 5

6. **D_INT[3]==1** → ***Output 2***
   To verify the use of this rule, all that had to be done was connect 24V to the digital input of the Interface, and check is the Output2 had been activated.

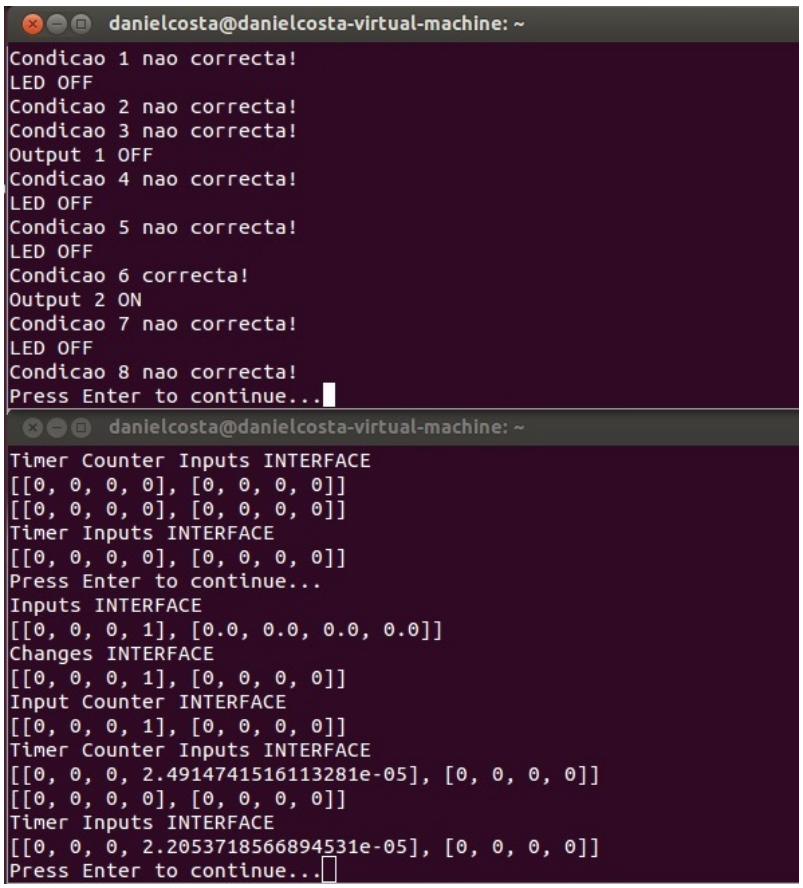


Figure 7.6: Result Condition 6

7. **A_INT[0]>3 → *LED***
   Connecting the third analogical input to a voltage divider using a potentiometer, and regulating the voltage first to 0V and then increasing it to 4V and back to 0V to see its behaviour. The Figure 7.7 shows that test.

```
Condicao 1 nao correcta!
LED OFF
Condicao 2 nao correcta!
Condicao 3 nao correcta!
Output 1 OFF
Condicao 4 nao correcta!
LED OFF
Condicao 5 nao correcta!
LED OFF
Condicao 6 nao correcta!
Output 2 OFF
Condicao 7 correcta!
LED ON
Condicao 8 nao correcta!
Press Enter to continue...
Condicao 1 nao correcta!
LED OFF
Condicao 2 nao correcta!
Condicao 3 nao correcta!
Output 1 OFF
Condicao 4 nao correcta!
LED OFF
Condicao 5 nao correcta!
LED OFF
Condicao 6 nao correcta!
Output 2 OFF
Condicao 7 nao correcta!
LED OFF
Condicao 8 nao correcta!
Press Enter to continue...
```

```
danielcosta@danielcosta-virtual-machine: ~
Inputs INTERFACE
[[0, 0, 0, 0], [3.9611110091209412, 0.0, 0.0, 0.0]]
Changes INTERFACE
[[0, 0, 0, 0], [0, 0, 0, 0]]
Input Counter INTERFACE
[[0, 0, 0, 0], [0, 0, 0, 0]]
Timer Counter Inputs INTERFACE
[[0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 0, 0, 1], [0, 0, 0, 0]]
Timer Inputs INTERFACE
[[0, 0, 0, 0], [0, 0, 0, 0]]
Press Enter to continue...
Inputs INTERFACE
[[0, 0, 0, 0], [0.39444442838430405, 0.0, 0.0, 0.0]]
Changes INTERFACE
[[0, 0, 0, 0], [-1, 0, 0, 0]]
Input Counter INTERFACE
[[0, 0, 0, 0], [0, 0, 0, 0]]
Timer Counter Inputs INTERFACE
[[0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 0, 0, 1], [0, 0, 0, 0]]
Timer Inputs INTERFACE
[[0, 0, 0, 0], [0, 0, 0, 0]]
```

Figure 7.7: Result Condition 7

8. **C_INT[0][3]>2,TC_INT[0][3]<20 → *email***
   This rule, like the rule number 5, is a counter rule. In Figure 7.8 we see the Input Counter of the Digital input number 4 (position 3) and the respective Timer Counter Input. The rule is True as the counter is 3 with the timer less than 20 seconds.

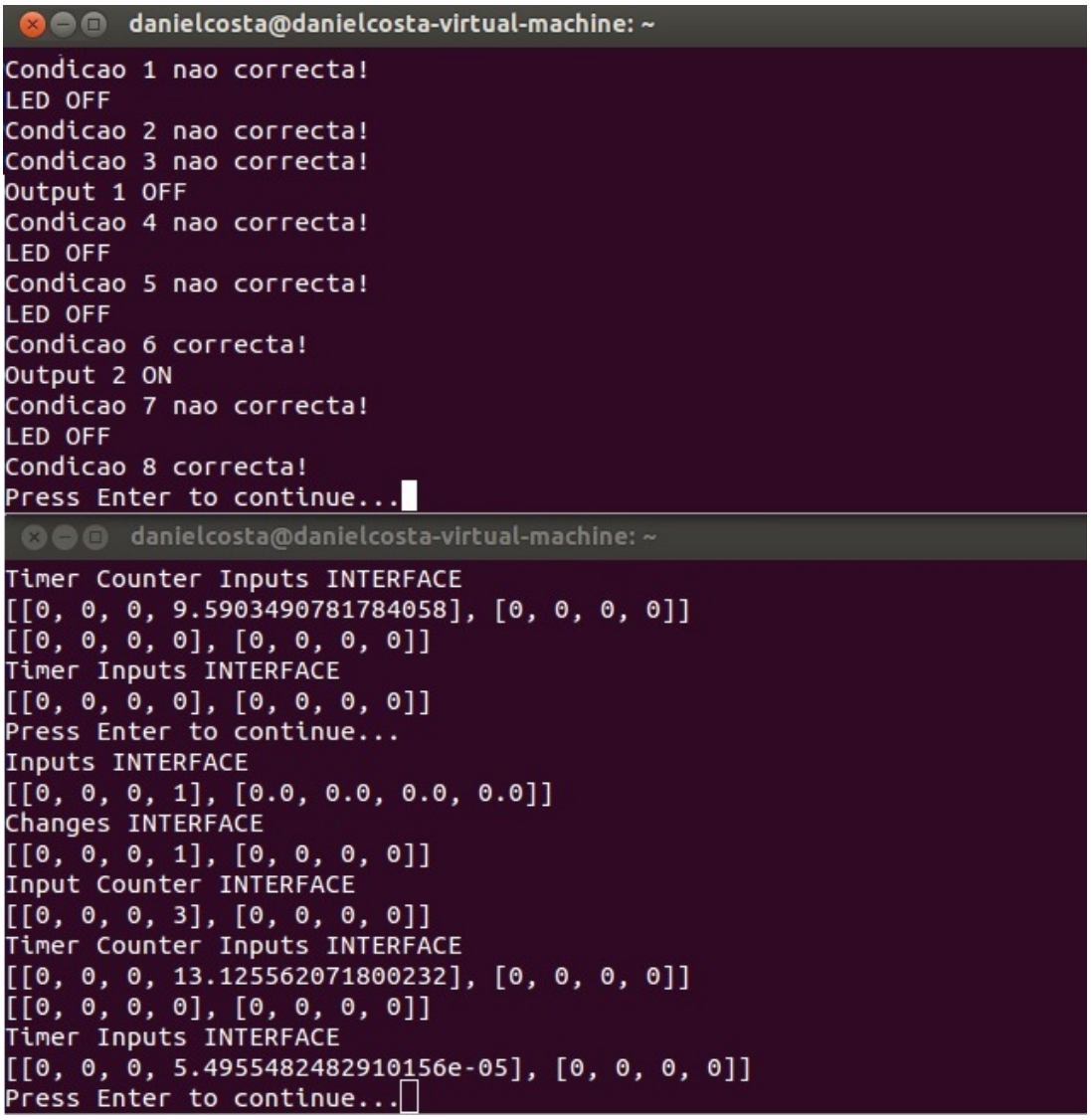


Figure 7.8: Result Condition 8

7.2 Conclusion

With these tests we can conclude that the main operations and requirements of the SO are fulfilled. Despite that, there are some bugs on the scripts that need to be fixed in future works. Bugs like:

- If several rules with the same output are created, the output may be powered off even if a rule is true;

- When scripts are running simultaneously and in infinite cycles, there are some problems with the database relation such as, trying to read and write at the same table on the same time;
- Can not execute both script at the same time on the BBB start up.

Despite these bugs the core functions of the SO developed in this project are fully operational.

Chapter 8

Conclusions and Future Work

This chapter covers the final conclusions of the project and it is divided in two parts:

- The first part where the final and major conclusions are exposed and debated, as well as the fulfilment of the goals purposed in the beginning;
- The second part where it is discussed the possible future works in order to improve this Smart Object.

8.1 Conclusions and Fulfilment of Goals

As we saw on this report, the work was divided in several parts.

In order to develop a solution for Smart Objects in the Manufacturing area, it was important to understand the problem, the motivation and the goals of the project.

It was conducted a survey concerning the technologies involved: Internet of Things and Smart Objects. It was easy to find a lot of information concerning the IoT, but not so much for Smart Objects since it is a recent subject in this area. The ADVENTURE Project was a crucial help in understanding the complex framework on a SO and all the platform surrounding it. A market survey was also conducted to be aware of the existing related products both full integrated Smart Objects solutions and component related products (such as sensoring interfaces or micro-processors).

Using this research it was easy to identify the requirements and functionalities for our Smart Object for Manufacturing, present in 3. The difficulty here was to know if those functionalities could be implemented and if they were be enough for that complex platform, difficulty that was surpassed along the project.

The necessity to divide the SO emerged and the results of that division are: sensoring, communication and processing.

On what concerns the first, sensoring, it was made a huge effort to have an interface capable of performing all the functionalities of the SO. After a intense market survey, a complete solution had not been found, so, and since it was an important part of the project, a new interface had to be made from scratch. After undertaking the concept and the main features of the I/O interface (digital and

analogic inputs and relay controlled outputs), it was time to make the schematics and build the PCB. Some difficulties appeared when connecting all the electronic features together in the same board, but were overcame with no major problems. At the execution of the schematics some new features needed to be added in order to the PCB and the SO fulfil all respective requirements. The final features of the I/O Interface are described at 4.2 and include, among others: digital (24V DC) and analogic(0-10V DC) inputs, relay controlled outputs and real-time clock.

The communication part was all about connecting the BeagleBone Black (BBB) to the PC and to the Omron PLC. The first one was achieved by Ethernet communication and used an Ethernet Switch to build a mini-network. This communication implied data exchanging and for that it was used Omron's FINS Protocol, through an existing program from INESC TEC PORTO that read the memory area of the PLC. With a study of the protocol manual, the area to read was changed to the input/output memory area.

As for the processing part, physically represented by the BBB, was responsible not only for interconnect all the parts but also to interpret the rules. The first part, concerning the communication part, was already handled and as for the sensoring part, that was physically connect by the BBB pins, a library (Adafruit GPIO) allowed the data recognition from the PCB. Regarding the rule interpretation, two Python scripts were created: one for collecting and processing the external data (inputs from PLC and Interface) and the other to read and interpret the condition and consecutive actions. A database is used to interconnect both scripts and its operation is much like a PLC program execution cycle: read the inputs, data processing, actualize the outputs and restarting. This being said, the scripts execution works like this:

- In inputs.py:
    1. Inputs read from PLC and Interface;
    2. Variables in inputs.py actualized (inputs, changes, counter and respective timers);
    3. Store variables on database in table logs and timers;
    4. Restart cycle.
- In rules.py:
    1. Fetch the rules from the database's rules table;
    2. Begin while cycle, each cycle for a rule;
    3. Variables collected from the logs and timer tables, according to the specific rule;
    4. Is the rule true or false?
    5. If true, perform respective action; if false, action not performed;
    6. Next rule.

Several difficulties appeared during the scripts writing, but they were surpassed with external help or by simple code debugging.

All these efforts and task, while the project execution, produced the results present in 7. The rules implemented were tested using the hardware and software available and those tests were very satisfying for the output of this dissertation. All conditions worked perfectly and the respective actions were successfully performed.

Since there were not any solutions for the sensoring part, the detail, concept and development of the solutions took longer than anticipated and delayed some possible future work with this solution. Despite that we can affirm that the main goal of this project was successfully fulfilled as this solution is capable of implementing Internet of Things features at a manufacturing machinery level as explained above.

On a general perspective, this was a very important step towards the Smart Object implementation in the Manufacturing area. The Smart Object is extremely well documented and we believe that anyone with some basic knowledge in the area is capable of analyse and understand the logical operation behind the object. For the ADVENTURE Project implementation, this SO is still lacking some features but the base is now created and in 8.2 it is described possible future works, based in this project.

8.2 Future Work

This project, due to its complexity and necessity of constant improvement, can develop some future works. For start there are some bugs in the python scripts that can be eliminated such as:

- If several conditions exist with the same action and if, for instance, the first is true and the second false, the output will be off;
- Some writing/reading problems with the database and the pickle file, that can be fixed using locks, this limiting the access to that files of one script at a time.

Being this a significant step on what concerns the application of the IoT in the manufacturing area, the innovation field is crucial and this can be improved by for example, endow the processing part of some operation process performance (OEE, energy consumption, etc. or adding the possibility to create and configure more complex rules (continuous or pattern based) than the ones already created (discrete). Some possible improvements also include the creation of an user interface for adding, editing and removing rules. A concept for this work was already created and is presented in D. Enabling more communication to this Smart Object, such as communication for a different PLC brand, since the FINS Protocol is only for Omron's PLCs and developing a different sensoring platform such as wireless sensor network would also be significant improvements to this solution.

Appendix A

ADVENTURE

In this chapter is it displayed an ADVENTURE user interface example.

A.1 User Interface

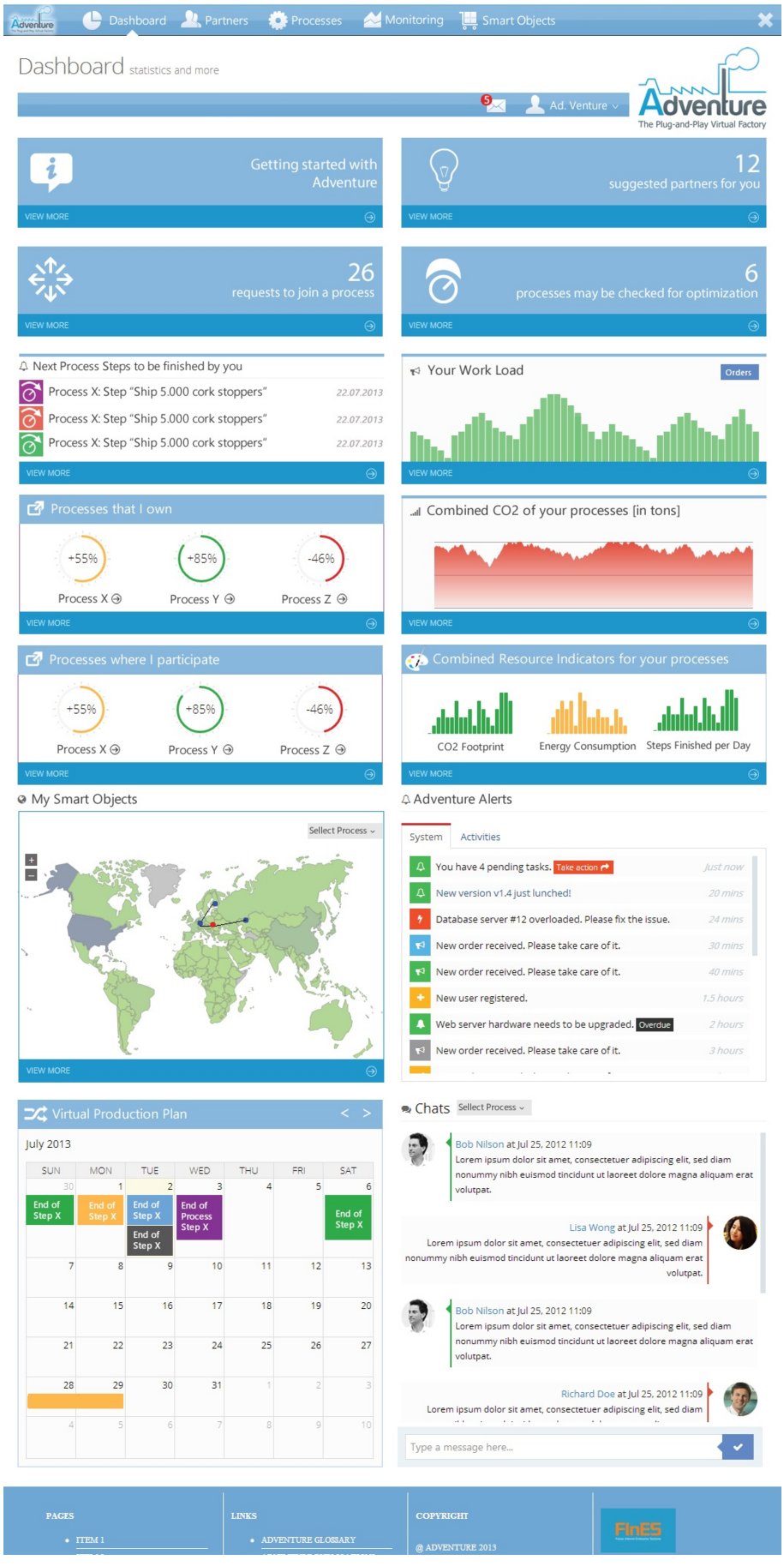


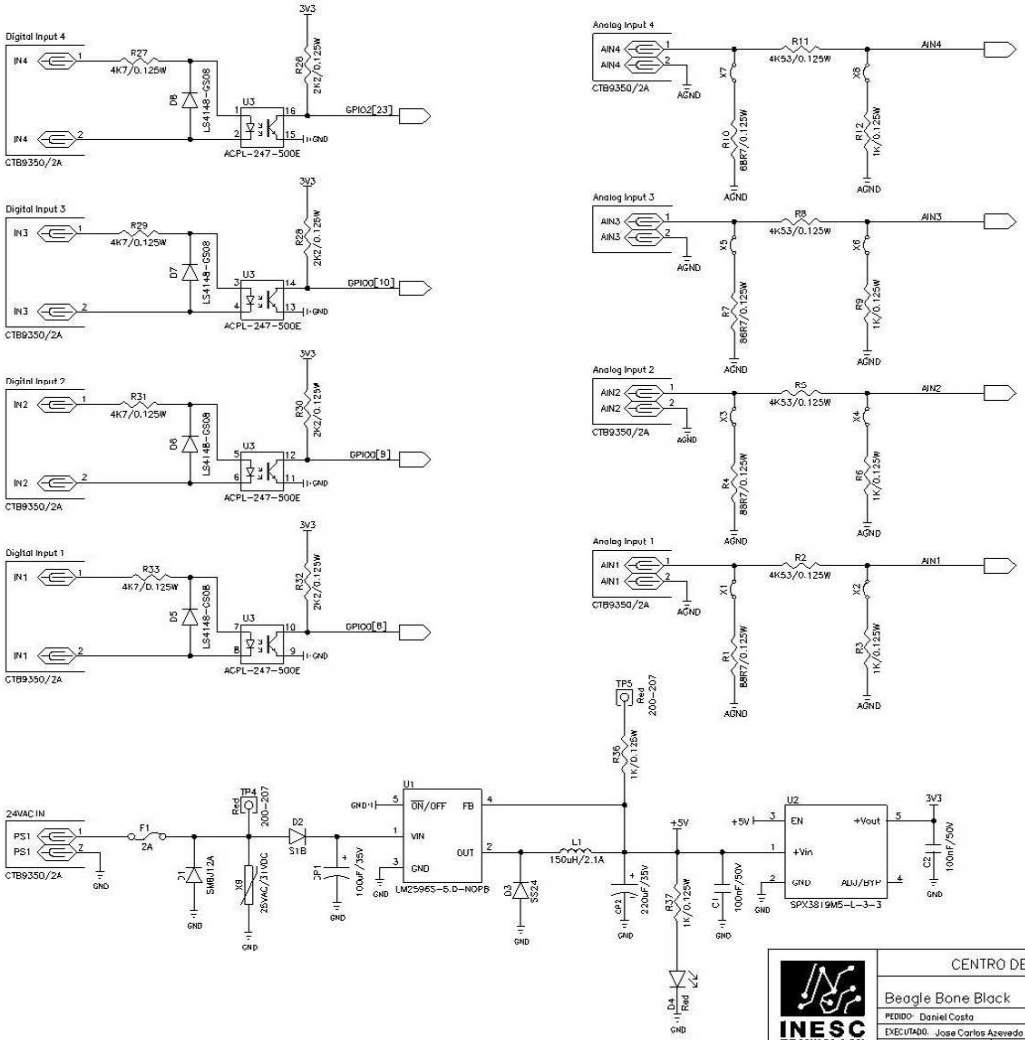
Figure A.1: ADVENTURE Goal [4]

Appendix B

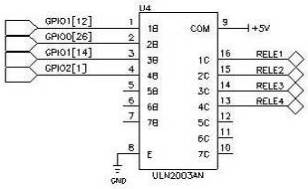
I/O Interface Schematics

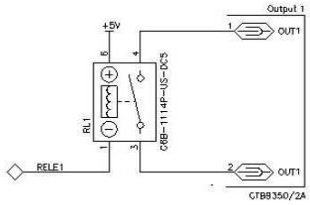
In this appendix are exposed the complete PCB schematics, for the sensoring part of the Smart Object present in 4.

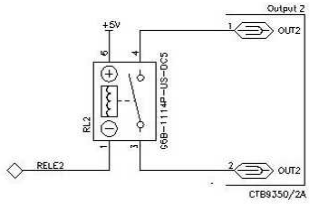
B.1 Printed Circuit Board

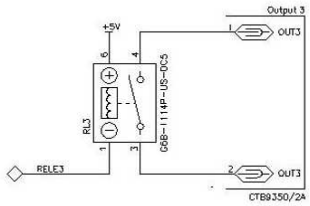


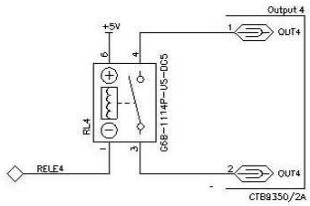
INESC TECHNOLOGY	CENTRO DE CAD PARA ELECTRONICA – PORTO		
	Beagle Bone Black	Inputs/PowerSupply	
	PEDIDO: Daniel Costa	DATA: 2014/04/1	
	EXECUTADO: Jose Carlos Azevedo	DATA: 2014/05/0	
	PROJ. No: ___-___-___	REQ. No: ___/____	FOLHA: 1 de 3 VERSAO: 1.0



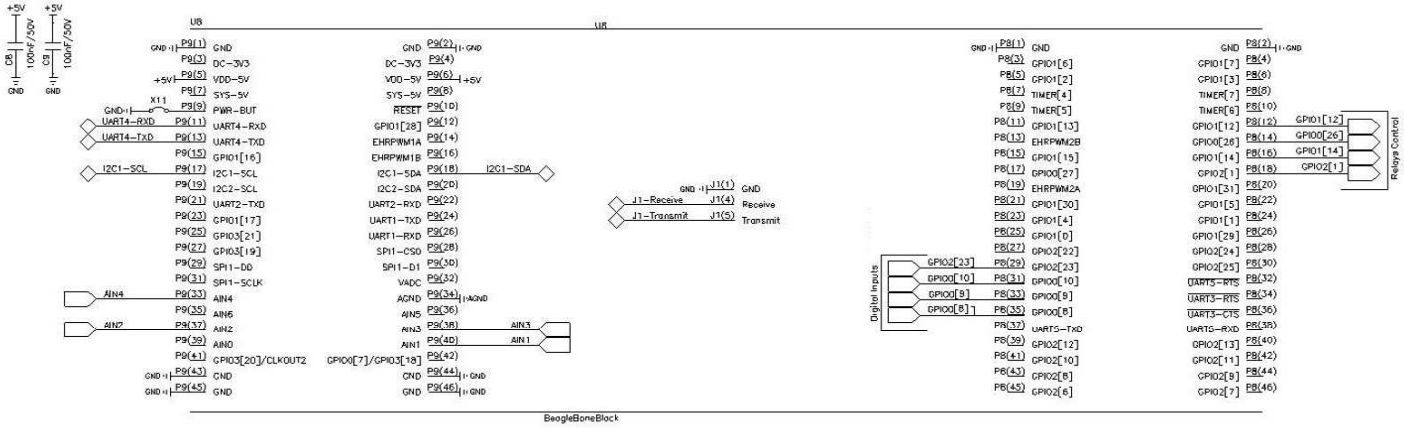


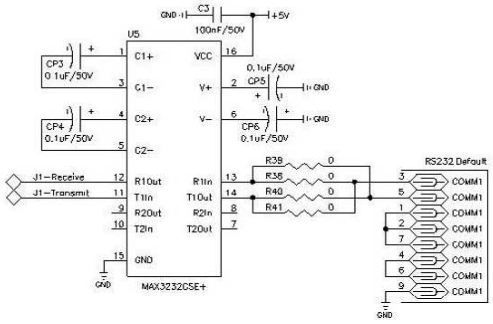


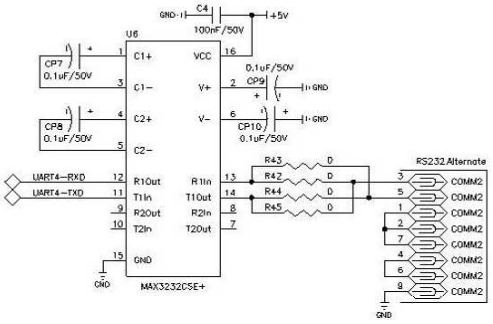


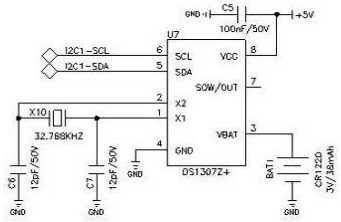


INESC TECHNOLOGY	CENTRO DE CAD PARA ELECTRONICA – PORTO			
	Beagle Bone Black	Outputs		
	PEDIDO: Daniel Costa		DATA:	2014/04/1
	EXECUTADO: Jose Carlos Azevedo		DATA:	2014/05/0
	PROJ. No: __._._.__	REQ. No: ___/____	FOLHA: 2 de 3	VERSAO: 1.0









INESC TECHNOLOGY	CENTRO DE CAD PARA ELECTRONICA – PORTO		
	Beagle Bone Black	uProcessor/COMM	
	PEDIDO: Daniel Costa		DATA: 2014/04/1
	EXECUTADO: Jose Carlos Azevedo		DATA: 2014/05/0
	PROJ. No: __.__.__	REQ. No: ___/____	FOLHA: 3 de 3 VERSAO: 1.0

Appendix C

Code

This appendix contains the full code of the communication protocol (present in chapter 5) and both python scripts for the processing component (related to chapter 6).

C.1 Omron Communication Code

C.1.1 finscommunicator.cpp

```
1  /*
2   * finscommunicator.cpp
3   *
4   *  Created on: 16 de Nov de 2012
5   *      Author: alvaro
6   */
7
8  #include "UDPSocket.h"
9  #include "finscommunicator.h"
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <iostream>
13 #include <iomanip>
14 #include <sstream>
15 #include <omp.h>
16 #include <stdio.h>
17 #include <stdlib.h>
18 #include <unistd.h>
19 #include <errno.h>
20 #include <string.h>
21 #include <netdb.h>
22 #include <sys/types.h>
23 #include <netinet/in.h>
24 #include <sys/socket.h>
25 #include <string>
26 #include <sstream>
27 #include <iostream>
```

```
28 #include <iomanip>
29 using namespace std;
30
31
32 finscommunicator::finscommunicator() {
33
34 }
35
36
37
38 bool finscommunicator::readFromPLC(std::string servAddress,unsigned short
       echoServPort,int startPos, int nrOfWordToRead,unsigned char memoryArea ){
39
40                 unsigned char fins_send_frame[18], fins_receive_frame[14+(
                       nrOfWordToRead*2)];
41
42
43                 stringstream nrOfWordToReadHEX;
44                 nrOfWordToReadHEX << setw(4) << setfill('0') << hex << (int)(
                       nrOfWordToRead);
45                 stringstream startPosHEX;
46                 startPosHEX << setw(4) << setfill('0') << hex << (int)(startPos
                       );
47
48
49                 fins_send_frame[0] = 0x80; /* ICF */
50                 fins_send_frame[1] = 0x00; /* RSV */
51                 fins_send_frame[2] = 0x02; /* GCT */
52                 fins_send_frame[3] = 0x00; /* DNA */
53                 fins_send_frame[4] = 0x01; /* DA1 */
54                 fins_send_frame[5] = 0x00; /* DA2 */
55                 fins_send_frame[6] = 0x00; /* SNA */
56                 fins_send_frame[7] = 0x64; /* SA1 */
57                 fins_send_frame[8] = 0x00; /* SA2 */
58                 fins_send_frame[9] = 0x00; /* SID */
59                 fins_send_frame[10] = 0x01; /* MRC */
60                 fins_send_frame[11] = 0x01; /* SRC */
61                 fins_send_frame[12] = memoryArea; /* VARIABLE TYPE: DM*/
62
63
64                 std::istringstream hex_chars_stream1(startPosHEX.str().substr
                       (0,2));
65                 unsigned int c1;
66                 hex_chars_stream1 >> std::hex >> c1;
67                 fins_send_frame[13] = static_cast<unsigned char>(c1);
68
69                 std::istringstream hex_chars_stream2(startPosHEX.str().substr
                       (2,4));
70                 unsigned int c2;
```

```
71                  hex_chars_stream2 >> std::hex >> c2;
72                  fins_send_frame[14] = static_cast<unsigned char>(c2);
73
74                  fins_send_frame[15] = 0x00;
75
76
77
78                  unsigned int c3;
79                  std::istringstream hex_chars_stream3(nrOfWordToReadHEX.str().
                        substr(0,2));
80                  hex_chars_stream3 >> std::hex >> c3;
81                  fins_send_frame[16] = static_cast<unsigned char>(c3);
82
83                  unsigned int c4;
84                  std::istringstream hex_chars_stream4(nrOfWordToReadHEX.str().
                        substr(2,4));
85                  hex_chars_stream4 >> std::hex >> c4;
86                  fins_send_frame[17] = static_cast<unsigned char>(c4);
87
88                  //[0002, 0301, 0000, 0001, 0000, 0000, 0314, 0100, 0087, 0003]
89                  string insert="";
90                  stringstream insertString;
91                    try {
92
93                              UDPSocket sock;
94                              // Send the string to the server
95                              sock.sendTo(fins_send_frame, 18, servAddress,
                                    echoServPort);
96                              // Receive a response
97                              sock.recv(fins_receive_frame, 14+(nrOfWordToRead*2)
                                    );
98                              //fins_receive_frame[respStringLen] = '\0';
                                                    // Terminate the string!
99                              unsigned int i = 0;
100                            /* cout << "Header: [";
101                             for (i = 0; i< 14; i++)
102                             {
103                                 if (i < 13)
104                                         cout << setw(2) << setfill('0') << hex
                                                << (int)( fins_receive_frame[i] )
                                                << " ";
105                                 else
106                                         cout << setw(2) << setfill('0') << hex
                                                << (int)( fins_receive_frame[i] );
107                                 //printf("%x ",fins_receive_frame[i]);
108                             }
109                             cout << "]" <<endl;*/
110
111                             int counter=(nrOfWordToRead*2)-1;
```

```
112                             int counterfinal=(nrOfWordToRead*2)-1;
113                             cout << "{\"I";
114                             cout << counterfinal-counter;
115                             cout << "\":\"";
116                             for (i = 14; i< sizeof(fins_receive_frame); i=i+2)
117                             {
118
119                                 cout << (int)(fins_receive_frame[i+1]);
120
121                                 counter--;
122                                         cout << "\",\"I";
123                                         cout << counterfinal-counter;
124                                         cout << "\":\"";
125
126                                 //printf("%x ",fins_receive_frame[i]);
127
128                                 cout << (int)(fins_receive_frame[i]);
129                                 if(i+1 < sizeof(fins_receive_frame)-1)
130                                         {
131                                                 counter--;
132                                                         cout << "\",\"I";
133                                                         cout << counterfinal-
                                                                counter;
134                                                         cout << "\":\"";
135
136                                         }
137                             }
138                             cout << "\"}";
139
140                           } catch (SocketException &e) {
141                             cerr << e.what() << endl;
142                             //exit(1);
143                           }
144
145          return true;
146
147 }
148
149
150 int finscommunicator::memory_area_designation_code(const int addr)
151 {
152
153         if(addr>0x0BFFF&&addr<0x0D800) //B0 , CIO area
154                 return 0xB0;
155         if(addr>0x0DDFF&&addr<0xE000) //work area
156                 return 0xB1; //TODO: clarified code , from memory area
                        designation codes table
157         if(addr>0xFFFF&&addr<0x18000) //DM area
158                 return 0x82;
```

```
159         if(addr>0xB9FF&&addr<0xBC00) //A448 - A959
160                 return 0xB3;
161         else
162                 return 0x80; //CIO,LR,HR,AR area*/
163 }
```

C.2 Processing Scripts

C.2.1 inputs.py

```
1  from flask import Flask
2  from flask.ext import restful
3  import subprocess
4  import sys
5  import os
6  import re
7  import Adafruit_BBIO.GPIO as GPIO
8  import sqlite3
9  import smtplib
10 import time
11 import pickle
12 import Adafruit_BBIO.ADC as ADC
13
14 global T
15 global T_counter
16 global G
17 global start_time
18
19 start_time = [[0 for col in range(8)] for row in range(4)]
20 start_time2 = [[0 for col in range(8)] for row in range(4)]
21 T = [[0 for col in range(8)] for row in range(4)]
22 T_counter = [[0 for col in range(8)] for row in range(4)]
23 G = [[0 for col in range(8)] for row in range(4)]
24
25
26 global T_INT
27 global T_counter_INT
28 global G_INT
29 global start_time_INT
30
31 start_time_INT = [[0 for col in range(4)] for row in range(2)]
32 start_time2_INT = [[0 for col in range(4)] for row in range(2)]
33 T_INT = [[0 for col in range(4)] for row in range(2)]
34 T_counter_INT = [[0 for col in range(4)] for row in range(2)]
35 G_INT = [[0 for col in range(4)] for row in range(2)]
36
37 #----------------------VARIAVEIS INTERFACE
      -----------------------------------#
38 DIN1="P8_29"
39 DIN2="P8_31"
40 DIN3="P8_33"
41 DIN4="P8_35"
42 AIN1='P9_40'
43 AIN2='P9_37'
```

```
44 AIN3='P9_38'
45 AIN4='P9_33'
46 OUTPUT1="P8_12"
47 OUTPUT2="P8_14"
48 OUTPUT3="P8_16"
49 OUTPUT4="P8_18"
50 #
       ------------------------------------------------------------------------#
51 #-------------------DEFINIR PINOS
       ----------------------------------------#
52 GPIO.cleanup()
53 GPIO.setup("P8_29", GPIO.IN)
54 GPIO.setup("P8_31", GPIO.IN)
55 GPIO.setup("P8_33", GPIO.IN)
56 GPIO.setup("P8_35", GPIO.IN)
57 ADC.setup()
58 #
       ------------------------------------------------------------------------#
59
60 def main():
61     plc=GetData()
62     while(1):
63         plc.get()
64         time.sleep(0.1)
65         #raw_input("Press Enter to continue...")
66
67 class GetData(object):
68         def db(self, number):
69 #             try:
70 #                 self.fp=open("shared.pkl","wb")
71 #             except IOError:
72             with open("shared.pkl",'rb') as fp:
73                 flag_PLC, flag_INT, flag_T_PLC, flag_T_INT, flag_C_PLC,
                       flag_C_INT = pickle.load(fp)
74             fp.close()
75
76             #LIGACAO A BASE DE DADOS
77             conn = sqlite3.connect("rules.db",timeout=10)
78             cursor = conn.cursor()
79             cursor.execute("SELECT id FROM logs")
80             result=cursor.fetchall()
81             leng=len(result)
82             input_id=leng+1
83             """==================================INPUTS
                   ========================================="""
84             """_____________________________INPUTS
                   PLC_____________________________________"""
85             #CONSTRUIR VECTOR DE INPUTS#
86             I=[]
87             j=0
88             while (j<len(number.keys())):
89                 i=0
90                 a=int(number['I%d' % j])
```

```
91                  b=format(a, '010b')
92                  c=b[2:]
93                  I.append([])
94
95                  while(i<len(c)):
96                      I[j].append(c[(len(c)-1)-i])
97                      i=i+1
98                  j=j+1
99              print 'Inputs'
100             print I
101             str_PLC = ','.join(str(e) for e in I)  #convert array of
                    inputs to string
102
103             """_____________________________INPUTS
                    INTERFACE_____________________________"""
104             #------------DIGITAL INPUTS-----------------
105             I_INT=[]
106             h=1
107             DIN=[]
108             d=29
109             while(h<5):
110                 f=GPIO.input("P8_%d" % d)
111                 if(f==0):
112                     value=1
113                 elif(f==1):
114                     value=0
115                 DIN.append(value)
116                 d=d+2
117                 h=h+1
118             I_INT.append(DIN)
119             #--------------ANALOG INPUTS-----------------
120             h=1
121             AIN=[]
122             while(h<5):
123                 ADC.read('AIN%d' % h)
124                 value = ADC.read('AIN%d' % h)
125                 voltage = value * 10 #1.8V
126                 AIN.append(voltage)
127                 h=h+1
128             I_INT.append(AIN) #vector com os inputs da interface
129 #             print 'Inputs INTERFACE'
130 #             print I_INT
131             str_INT = ','.join(str(e) for e in I_INT)
132             """================================/INPUTS
                    /====================================="""
133             #
                    ##########################################################################

134             """================================CHANGES
                    ====================================="""
135             """_____________________________CHANGES
                    PLC_____________________________"""
136             if(leng==0):
137                 A=[[0 for col in range(8)] for row in range(4)]
138             else:
```

```
139                     cursor.execute("SELECT input_PLC FROM logs WHERE id=?",(
                            leng,))
140                     x=cursor.fetchone()
141                     y=''.join(x)
142                     A=eval(y)
143                 n=len(I)
144                 f=0
145                 J=[]
146                 changes_PLC=0
147                 while(f<n):
148                     b=0
149                     J.append([])
150                     while(b<8):
151                         if (int(A[f][b])<int(I[f][b])):
152                             J[f].append(1)
153                             changes_PLC=1
154                         elif (int(A[f][b])>int(I[f][b])):
155                             J[f].append(-1)
156                             changes_PLC=1
157                         else:
158                             J[f].append(0)
159                         b=b+1
160                     f=f+1
161                 print 'Changes'
162                 print J
163                 str_changes_PLC = ','.join(str(e) for e in J)  #convert array
                        of changes to string
164                 """_____________________________CHANGES
                        INTERFACE_____________________________________"""
165                 if(leng==0):
166                     B=[[0 for col in range(4)] for row in range(2)]
167                 else:
168                     cursor.execute("SELECT input_INT FROM logs WHERE id=?",(
                            leng,))
169                     x=cursor.fetchone()
170                     if(x[0]==None):
171                         B=[[0 for col in range(4)] for row in range(2)]
172                     else:
173                         y=''.join(x)
174                         B=eval(y)
175                 f=0
176                 J_INT=[]
177                 changes_INT=0
178                 while(f<2):
179                     b=0
180                     J_INT.append([])
181                     while(b<4):
182                         if (int(B[f][b])<int(I_INT[f][b])):
183                             J_INT[f].append(1)
184                             changes_INT=1
185                         elif (int(B[f][b])>int(I_INT[f][b])):
186                             J_INT[f].append(-1)
187                             changes_INT=1
188                         else:
189                             J_INT[f].append(0)
```

```
190                     b=b+1
191                 f=f+1
192 #               print 'Changes INTERFACE'
193 #               print J_INT
194             str_changes_INT= ','.join(str(e) for e in J_INT)   #convert
                    array of changes to string
195             """==================================CONTADOR
                    ====================================="""
196             """_____________________________________CONTADOR
                    PLC_____________________________________"""
197             f=0
198             while(f<n):
199                 b=0
200                 while(b<8):
201                     if(flag_C_PLC[f][b]):
202                         if(J[f][b]==1):
203                             G[f][b]=G[f][b]+1
204                     b=b+1
205                 f=f+1
206             print 'Input Counter'
207             print G
208             str_counter_PLC = ','.join(str(e) for e in G)
209             f=0
210             while(f<n):
211                 b=0
212                 while(b<8):
213                     if(flag_T_PLC[f][b]):
214                         if(start_time2[f][b]==0 and J[f][b]==1):
215                             start_time2[f][b] = time.time()
216                         if(start_time2[f][b]==0):
217                             T_counter[f][b]=0
218                         else:
219                             T_counter[f][b]=(time.time() - start_time2[f
                                ][b])
220                     b=b+1
221                 f=f+1
222             print 'Timer Counter Inputs'
223             print T_counter
224             timestr2 = ','.join(str(e) for e in T_counter)
225
226             changes_counter_PLC=0
227             coun=''
228             counT=''
229             f=0
230             while(f<n):
231                 b=0
232                 while(b<8):
233                     if(flag_PLC[f][b]==1):
234                         changes_counter_PLC=1
235                         cursor.execute("SELECT counter_PLC FROM logs
                                WHERE id=(SELECT MAX(id) FROM logs)")
236                         x=cursor.fetchone()
237                         y=''.join(x)
238                         coun=eval(y)
239                         coun[f][b]=0
```

```
240                         G[f][b]=0
241                         cursor.execute("SELECT timer_counter_PLC FROM
                                timers WHERE id=(SELECT MAX(id) FROM timers)"
                                )
242                         x=cursor.fetchone()
243                         y=''.join(x)
244                         counT=eval(y)
245                         counT[f][b]=0
246                         start_time2[f][b]=0
247                         flag_PLC[f][b]=0
248                     b=b+1
249                 f=f+1
250             if(changes_counter_PLC):
251                 print 'zeros'
252                 str_counter_PLC=','.join(str(e) for e in coun)
253                 timestr2=','.join(str(e) for e in counT)
254 
255             """_____CONTADOR
                    INTERFACE_____"""
256             f=0
257             while(f<2):
258                 b=0
259                 while(b<4):
260                     if(flag_C_INT[f][b]):
261                         if(J_INT[f][b]==1):
262                             G_INT[f][b]=G_INT[f][b]+1
263                     b=b+1
264                 f=f+1
265 #             print 'Input Counter INTERFACE'
266 #             print G_INT
267             str_counter_INT = ','.join(str(e) for e in G_INT)
268 
269             f=0
270             while(f<2):
271                 b=0
272                 while(b<4):
273                     if(flag_T_INT[f][b]):
274                         if(start_time2_INT[f][b]==0 and G_INT[f][b]==1):
275                             start_time2_INT[f][b] = time.time()
276                         if(start_time2_INT[f][b]==0):
277                             T_counter_INT[f][b]=0
278                         else:
279                             T_counter_INT[f][b]=(time.time() -
                                    start_time2_INT[f][b])
280                     b=b+1
281                 f=f+1
282 #             print 'Timer Counter Inputs INTERFACE'
283 #             print T_counter_INT
284             timestr2_INT = ','.join(str(e) for e in T_counter_INT)
285 
286             changes_counter_INT=0
287             coun=''
288             counT=''
289             #print flag_INT
290             f=0
```

```
291                 while(f<2):
292                     b=0
293                     while(b<4):
294                         if(flag_INT[f][b]==1):
295                             changes_counter_INT=1
296                             cursor.execute("SELECT counter_INT FROM logs
                                    WHERE id=(SELECT MAX(id) FROM logs)")
297                             x=cursor.fetchone()
298                             y=''.join(x)
299                             coun_INT=eval(y)
300                             coun_INT[f][b]=0
301                             G_INT[f][b]=0
302                             cursor.execute("SELECT timer_counter_INT FROM
                                    timers WHERE id=(SELECT MAX(id) FROM timers)"
                                    )
303                             x=cursor.fetchone()
304                             y=''.join(x)
305                             counT_INT=eval(y)
306                             counT_INT[f][b]=0
307                             start_time2_INT[f][b]=0
308                             flag_INT[f][b]=0
309                         b=b+1
310                     f=f+1
311                 if(changes_counter_INT):
312                     str_counter_INT=','.join(str(e) for e in coun_INT)
313                     timestr2_INT=','.join(str(e) for e in counT_INT)
314
315                 """==============================/CONTADOR
                        /====================================="""
316                 #ESCREVER NA BASE DE DADOS#
317                 datime=time.strftime('%Y-%m-%d %H:%M:%S')
318                 if(changes_PLC or changes_INT or changes_counter_PLC or
                        changes_counter_INT):
319                     cursor.execute("INSERT INTO logs VALUES (?,?,?,?,?,?,?,?)
                            ",(input_id,str_PLC,str_changes_PLC,str_counter_PLC,
                            str_INT,str_changes_INT,str_counter_INT,datime))
320                     conn.commit()
321
322                 """==============================/CHANGES
                        /====================================="""
323                 #
                        ##########################################################################

324                 """==================================TIMER
                        ====================================="""
325                 """_____________________________TIMER
                        PLC_____________________________________"""
326                 f=0
327                 while(f<n):
328                     b=0
329                     while(b<8):
330                         if(int(J[f][b])==1 and int(I[f][b])==1):
331                             start_time[f][b] = time.time()
332                         elif(int(J[f][b])==-1):
333                             start_time[f][b]=0
```

```
334
335                     if(int(I[f][b])==1):
336                         T[f][b]=(time.time() - start_time[f][b])
337
338                     else:
339                         T[f][b]=0
340                     b=b+1
341                 f=f+1
342             print 'Timer Inputs'
343             print T
344             timestr= ','.join(str(e) for e in T)  #convert array of
                    timers to string
345             """_____________________________TIMER
                    INTERFACE_____________________________________"""
346             f=0
347             while(f<2):
348                 b=0
349                 while(b<4):
350                     if(int(J_INT[f][b])==1 and int(I_INT[f][b])==1):
351                         start_time_INT[f][b] = time.time()
352                     elif(int(J_INT[f][b])==-1):
353                         start_time_INT[f][b]=0
354
355                     if(int(I_INT[f][b])==1):
356                         T_INT[f][b]=(time.time() - start_time_INT[f][b])
357
358                     else:
359                         T_INT[f][b]=0
360                     b=b+1
361                 f=f+1
362 #               print 'Timer Inputs INTERFACE'
363 #               print T_INT
364             timestr_INT= ','.join(str(e) for e in T_INT)  #convert array
                    of timers to string
365             """_____________________ESCREVER NA BASE DE
                    DADOS_____________________________"""
366
367             cursor.execute("SELECT id FROM timers")      #ID do timer
368             result=cursor.fetchall()                     #ID do timer
369             leng=len(result)                             #ID do timer
370             timer_id=leng+1                              #ID do timer
371
372             cursor.execute("INSERT INTO timers VALUES (?,?,?,?,?)",(
                    timer_id,timestr,timestr2,timestr_INT,timestr2_INT))
373             conn.commit()
374             """==================================/TIMER
                    /======================================="""
375         #GET DATA FROM PLC
376         def get(self):
377             args = ("./omron")
378             popen = subprocess.Popen(args, stdout=subprocess.PIPE)
379             popen.wait()
380             output = popen.stdout.read()
381             a=eval(output)
382
```

```
383             self.db(a)
384
385 if __name__ == "__main__":
386     main()
```

C.2.2 rules.py

```
1  from flask import Flask
2  from flask.ext import restful
3  import subprocess
4  import sys
5  import os
6  import re
7  import Adafruit_BBIO.GPIO as GPIO
8  import sqlite3
9  import smtplib
10 import pickle
11 import time
12 import subprocess
13
14 #subprocess.Popen("python inputs.py 1", shell=True)
15
16 F=[]
17 #--------------------VARIAVEIS INTERFACE
       ------------------------------------------#
18 DIN1="P8_29"
19 DIN2="P8_31"
20 DIN3="P8_33"
21 DIN4="P8_35"
22 AIN1="P9_40"
23 AIN2="P9_37"
24 AIN4="P9_33"
25 AIN3="P9_38"
26 'OUTPUT1','P8_12'
27 'OUTPUT2','P8_14'
28 'OUTPUT3','P8_16'
29 'OUTPUT4','P8_18'
30 #
       ------------------------------------------------------------------------#

31 global flag_PLC
32 global flag_INT
33 global flag_T_PLC
34 global flag_T_INT
35 global flag_C_PLC
36 global flag_C_INT
37 flag_PLC=[[0 for col in range(8)] for row in range(4)]
38 flag_T_PLC=[[0 for col in range(8)] for row in range(4)]
39 flag_C_PLC=[[0 for col in range(8)] for row in range(4)]
40 flag_INT=[[0 for col in range(4)] for row in range(2)]
41 flag_T_INT=[[0 for col in range(4)] for row in range(2)]
42 flag_C_INT=[[0 for col in range(4)] for row in range(2)]
43 #-------------------DEFINIR PINOS
       ------------------------------------------#
44 GPIO.setup('P8_12', GPIO.OUT)
```

```
45 GPIO.setup('P8_14', GPIO.OUT)
46 GPIO.setup('P8_16', GPIO.OUT)
47 GPIO.setup('P8_18', GPIO.OUT)
48 #
       ------------------------------------------------------------------------#
49
50 #app = Flask(__name__)
51 #api = restful.Api(app)
52
53 def main():
54     plc=GetRules()
55     while(1):
56         plc.get()
57         #raw_input("Press Enter to continue...")
58         """--------------------RESET flags--------------------"""
59         flag_PLC=[[0 for col in range(8)] for row in range(4)]
60         flag_T_PLC=[[0 for col in range(8)] for row in range(4)]
61         flag_C_PLC=[[0 for col in range(8)] for row in range(4)]
62         flag_INT=[[0 for col in range(4)] for row in range(2)]
63         flag_T_INT=[[0 for col in range(4)] for row in range(2)]
64         flag_C_INT=[[0 for col in range(4)] for row in range(2)]
65
66 class GetRules(restful.Resource):
67
68         def rules(self,number):
69             """---------------------------INPUTS
                    ----------------------------------------------------"""
70             conn = sqlite3.connect("rules.db", timeout=10)
71             cursor = conn.cursor()
72             cursor.execute("SELECT input_PLC FROM logs WHERE id=(SELECT
                    MAX(id) FROM logs)")
73             x=cursor.fetchone()
74             y=''.join(x)
75             I_PLC=eval(y)#I_PLC
76             cursor.execute("SELECT input_INT FROM logs WHERE id=(SELECT
                    MAX(id) FROM logs)")
77             x=cursor.fetchone()
78             y=''.join(x)
79             I_INT=eval(y)#I_INT
80             D_INT=I_INT[0]#Digital Inputs
81             A_INT=I_INT[1]#Analog Inputs
82             #-----------------DATABASE-------------------#
83             cursor.execute("SELECT * FROM rules")
84             result=cursor.fetchall()
85             leng=len(result)
86             i=0
87             """-------------------------------------CONDICAO/PROCESSING
                    ------------------------------------------------"""
88
89             while(i<leng):
90                 res=result[i]
91                 idd=res[0]
92                 condicao=res[1]
93                 s_n=0
```

```
94                  s_counter=[0,0]
95                  s_counter_INT=[0,0]
96                  special=0
97
98                  if 'C_PLC' in condicao:#C_PLC
99                      special=1
100                     cursor.execute("SELECT counter_PLC FROM logs WHERE id
                            =(SELECT MAX(id) FROM logs)")
101                     x=cursor.fetchone()
102                     w=''.join(x)
103                     C_PLC=eval(w)#C_PLC
104
105                     cursor.execute("SELECT timer_counter_PLC FROM timers
                            WHERE id=(SELECT MAX(id) FROM timers)")
106                     x=cursor.fetchone()
107                     w=''.join(x)
108                     TC_PLC=eval(w)#TC_PLC
109
110
111                 if 'C_INT' in condicao:#C_INT
112                     special=1
113                     cursor.execute("SELECT counter_INT FROM logs WHERE id
                            =(SELECT MAX(id) FROM logs)")
114                     x=cursor.fetchone()
115                     w=''.join(x)
116                     C_INT=eval(w)#C_INT
117
118                     cursor.execute("SELECT timer_counter_INT FROM timers
                            WHERE id=(SELECT MAX(id) FROM timers)")
119                     x=cursor.fetchone()
120                     w=''.join(x)
121                     TC_INT=eval(w)#TC_INT
122
123
124                 if 'T_PLC' in condicao:#T_PLC
125                     cursor.execute("SELECT timer_PLC FROM timers WHERE id
                            =(SELECT MAX(id) FROM timers)")
126                     x=cursor.fetchone()
127                     w=''.join(x)
128                     T_PLC=eval(w)#T_PLC
129
130                 if 'TC_PLC' in condicao:
131                     special=1
132                     y=condicao.split(',')
133                     z=re.findall(r'\d+', y[0])
134                     e=int(z[0])
135                     d=int(z[1])
136                     flag_T_PLC[e][d]=1
137
138                 if 'T_INT' in condicao:#T_INT
139                     cursor.execute("SELECT timer_INT FROM timers WHERE id
                            =(SELECT MAX(id) FROM timers)")
140                     x=cursor.fetchone()
141                     w=''.join(x)
142                     T_INT=eval(w)#T_INT
```

```
143
144                 if 'TC_INT' in condicao:
145                     special=1
146                     y=condicao.split(',')
147                     z=re.findall(r'\d+', y[0])
148                     e=int(z[0])
149                     d=int(z[1])
150                     flag_T_INT[e][d]=1
151
152                 if 'C_PLC' in condicao:#C_PLC
153                     s_counter=eval(condicao)
154                     y=condicao.split(',')
155                     z=re.findall(r'\d+', y[1])
156                     g=int(z[0])
157                     h=int(z[1])
158                     flag_C_PLC[g][h]=1
159                     #print s_counter
160                     if (not (s_counter[1])):
161                         #print 'flag'
162                         flag_PLC[g][h]=1
163                     else:
164                         flag_PLC[g][h]=0
165
166                 if 'C_INT' in condicao:#C_INT
167                     s_counter_INT=eval(condicao)
168                     y=condicao.split(',')
169                     z=re.findall(r'\d+', y[1])
170                     g=int(z[0])
171                     h=int(z[1])
172                     flag_C_INT[g][h]=1
173                     #print s_counter_INT
174                     if (not (s_counter_INT[1])):
175                         #print 'flag int'
176                         flag_INT[g][h]=1
177                     else:
178                         flag_INT[g][h]=0
179
180                 if(not special):
181                     s_n=eval(condicao) #compara a condicao e retorna true
                            se igual e falso se diferente
182
183 #                   try:
184 #                       self.fp=open("shared.pkl","wb")
185 #                   except IOError:
186                     with open("shared.pkl",'wb') as fp:
187                         pickle.dump([flag_PLC, flag_INT, flag_T_PLC,
                            flag_T_INT, flag_C_PLC, flag_C_INT], fp)
188                     fp.close()
189
190                     acao=[0,0]
191                     if(s_n or (s_counter[0] and s_counter[1]) or (
                        s_counter_INT[0] and s_counter_INT[1])):
192                         acao[0]=1
193                         acao[1]=res[2]
194                         print 'Condicao %d correcta!' % idd
```

```
195                     self.action(acao)
196                 else:
197                     acao[0]=0
198                     acao[1]=res[2]
199                     print 'Condicao %d nao correcta!' % idd
200                     self.action(acao)
201 
202                 i=i+1
203 
204         """-----------------------------------ACAO/OUTPUT
                ----------------------------------------------"""
205         def action(self,acao):
206 
207             if (acao[0]==0):
208                 if(acao[1]=='LED'):
209                     print 'LED OFF'
210                 elif (acao[1]=='1' or acao[1]=='2' or acao[1]=='3' or
                        acao[1]=='4'):
211                     output=int(acao[1])
212                     print 'Output %d OFF' % output
213 
214                     if(acao[1]=='1'):
215                         GPIO.output(('P8_12'),GPIO.LOW)
216                     elif(acao[1]=='2'):
217                         GPIO.output(('P8_14'),GPIO.LOW)
218                     elif(acao[1]=='3'):
219                         GPIO.output(('P8_16'),GPIO.LOW)
220                     elif(acao[1]=='4'):
221                         GPIO.output(('P8_18'),GPIO.LOW)
222             elif(acao[0]==1):
223                 if(acao[1]=='LED'):
224                     print 'LED ON'
225 
226                 if(acao[1]=='email'):
227                     print 'Enviar Email!'
228                     """
229                     import smtplib
230 
231                     fromaddr = 'ee09173fe.up.pt'toaddrs =
                            'ee09173fe.up.pt'
232                     msg = 'There was a terrible error that occured and I
                            wanted you to know!'
233 
234                     # Credentials (if needed)
235                     username = 'ee09173'
236                     password = '**********'
237 
238                     # The actual mail send
239                     server = smtplib.SMTP('smtp.fe.up.pt:25')
240                     server.starttls()
241                     server.login(username,password)
242                     server.sendmail(fromaddr, toaddrs, msg)
243                     server.quit()
244                     """
245 
```

```
246                 if (acao[1]=='1' or acao[1]=='2' or acao[1]=='3' or acao
                        [1]=='4'):
247                     output=int(acao[1])
248                     print 'Output %d ON' % output
249
250                     if(acao[1]=='1'):
251                         GPIO.output(('P8_12'),GPIO.HIGH)
252                     elif(acao[1]=='2'):
253                         GPIO.output(('P8_14'),GPIO.HIGH)
254                     elif(acao[1]=='3'):
255                         GPIO.output(('P8_16'),GPIO.HIGH)
256                     elif(acao[1]=='4'):
257                         GPIO.output(('P8_18'),GPIO.HIGH)
258
259
260 #api.add_resource(GetRules, '/')
261
262 if __name__ == '__main__':
263         #app.run(debug=True,host="192.168.250.100")
264         main()
```

Appendix D

User Visual Interface

In this chapter it is presented a concept of a possible user interface implementation. Each section represents the respective page of the interface.

D.1 Inputs

The inputs page is represented in Figure D.7. The inputs are actualized as they are in the script, making this a real-time application. The color code is green is active and red if not.

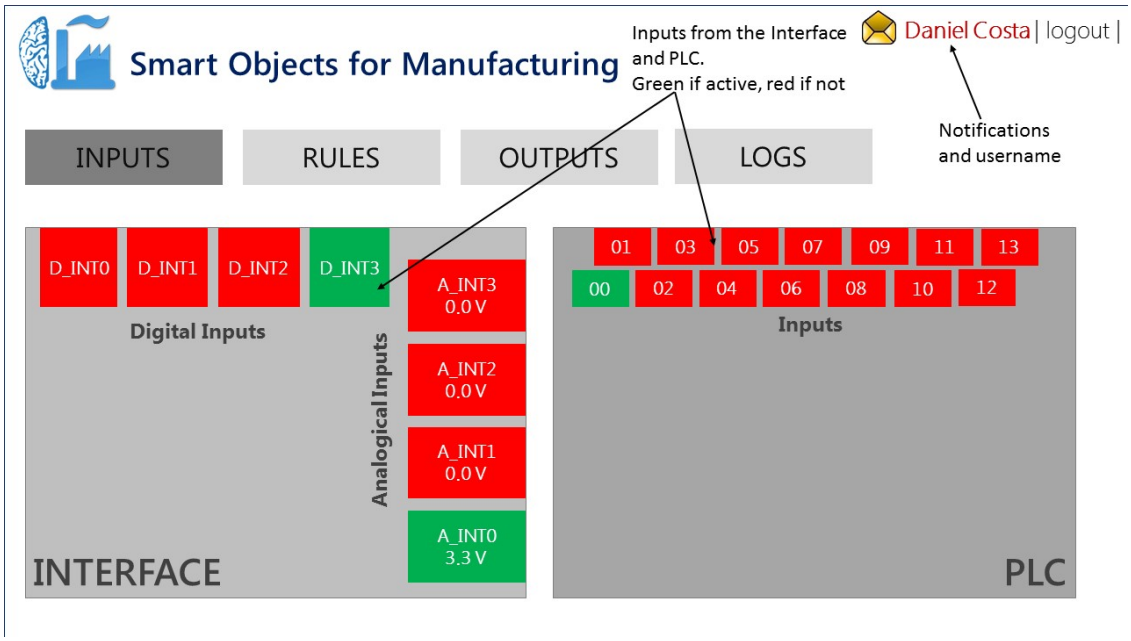


Figure D.1: Inputs Interface Screen

D.2 Rules

The rules screen as all the rules represented with the same color code as the inputs, as shown in Figure D.2. It is possible to edit, remove or add new rules, as demonstrated in the use case on D.2.1.

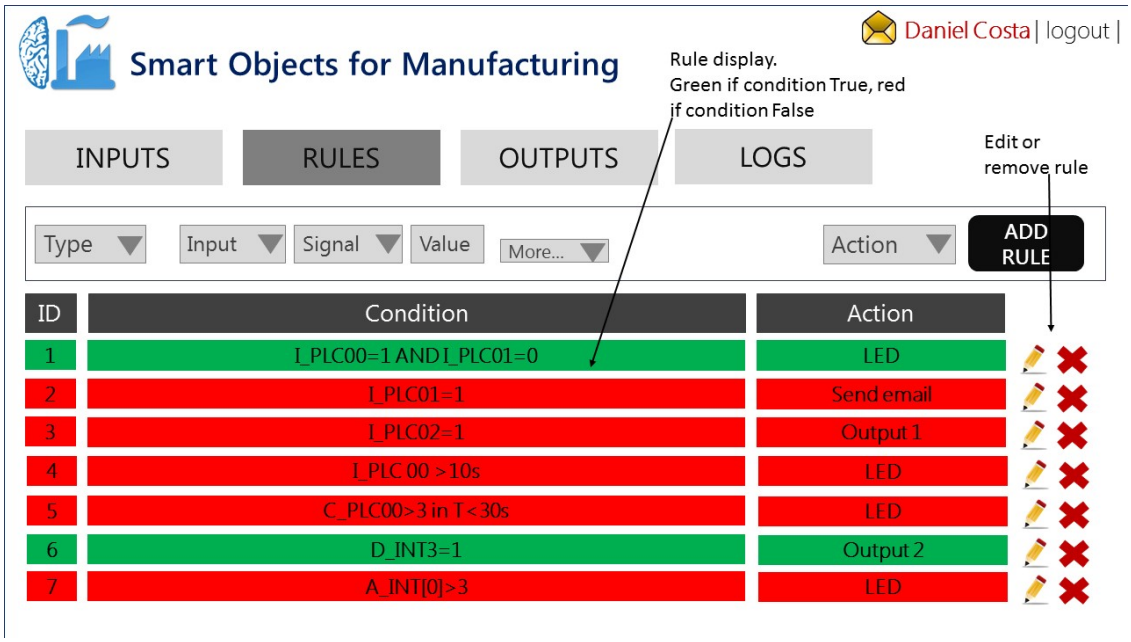


Figure D.2: Rules Interface Screen

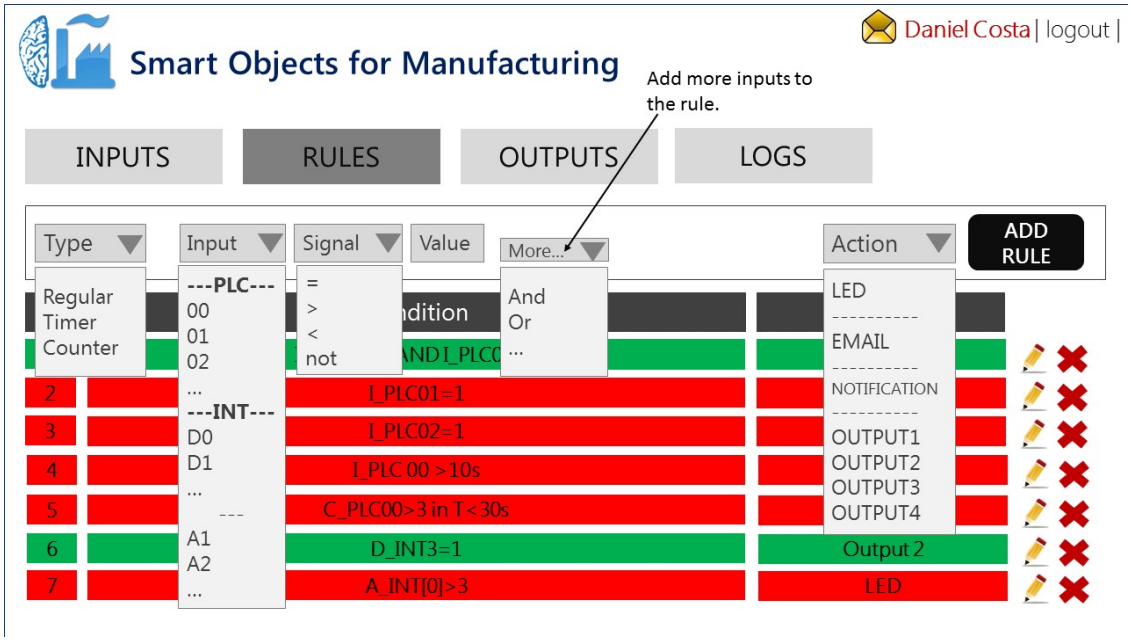


Figure D.3: Possibilities on adding a new rule

D.2.1 Use Case

In this use case, the user in adding a new rule. The Figure D.4 and D.5 show the user adding a counter rule:

1. Select the rule type - **Counter**;
2. Select the input to be used - **D_INT3**;
3. The signal to use - **<**;
4. Editing the value next to the signal - **2**;
5. Entering the time limit of the condition - **20 seconds**;
6. Select the action to perform if condition is true - **EMAIL**;
7. Click in **"ADD RULE"**.

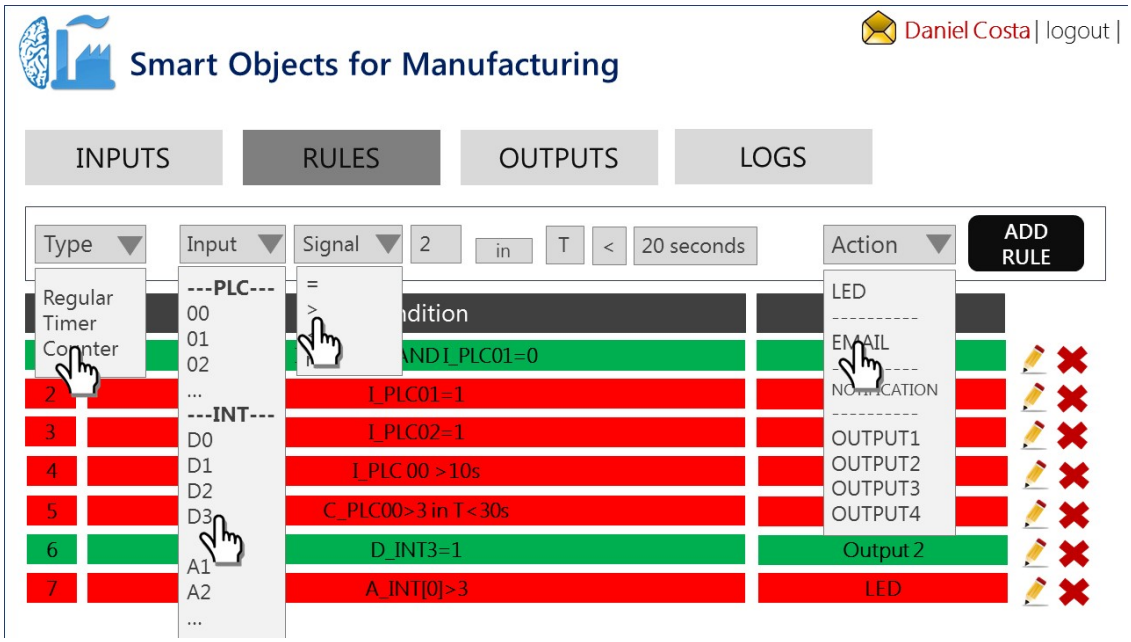


Figure D.4: Interface Use case - Adding a rule

So now the user as created the rule that is True when the digital input is activated more than two times in twenty seconds, shown in Figure D.5.

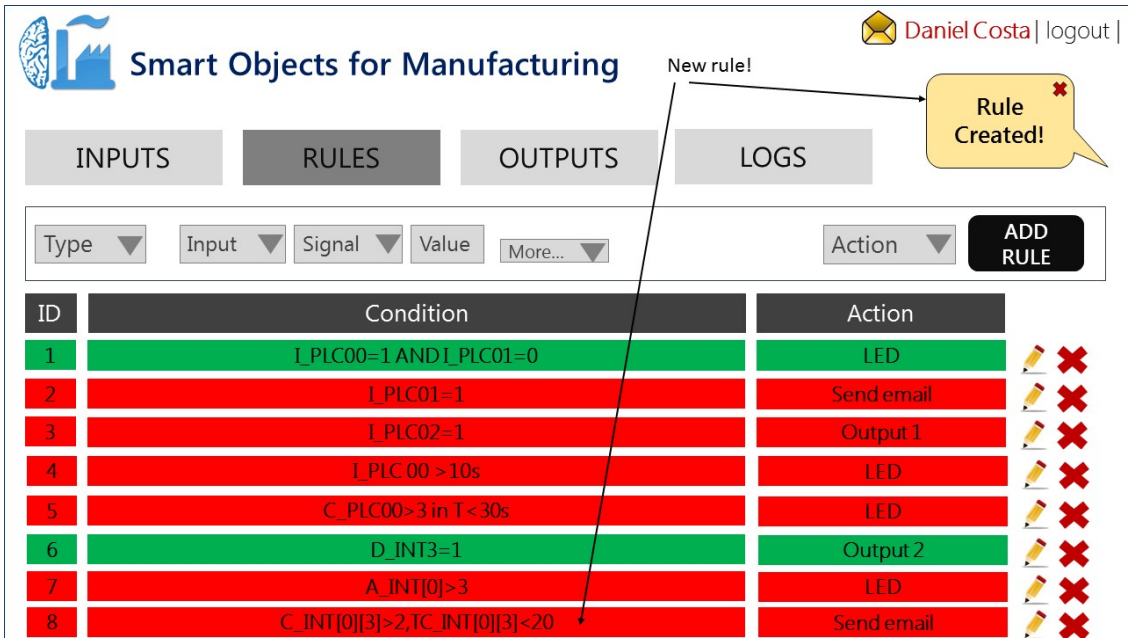


Figure D.5: Interface Use Case - Rule created

D.3 Outputs

The outputs screen, much like the Inputs page, has the interface and the outputs with the color code specified. Beside that, allows the user to configure the email service and the LED, as Figure D.6 shows.

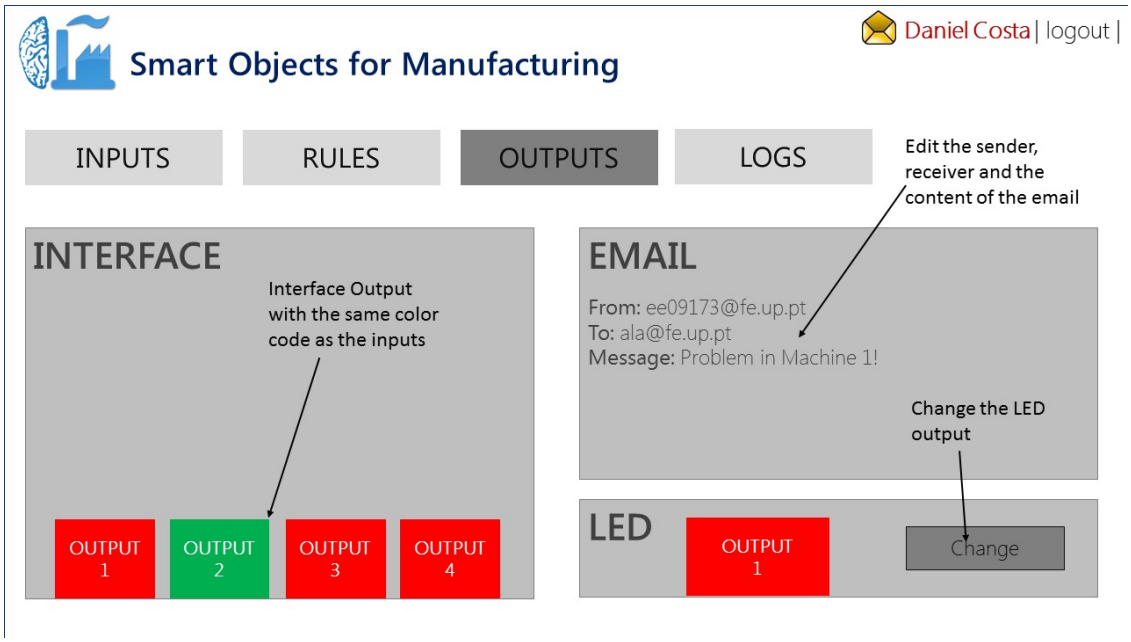


Figure D.6: Outputs Interface Screen

D.4 Logs

The logs screen is an easy way to consult the logs table of the database. All the information on the table was already presented in 6.3.2.

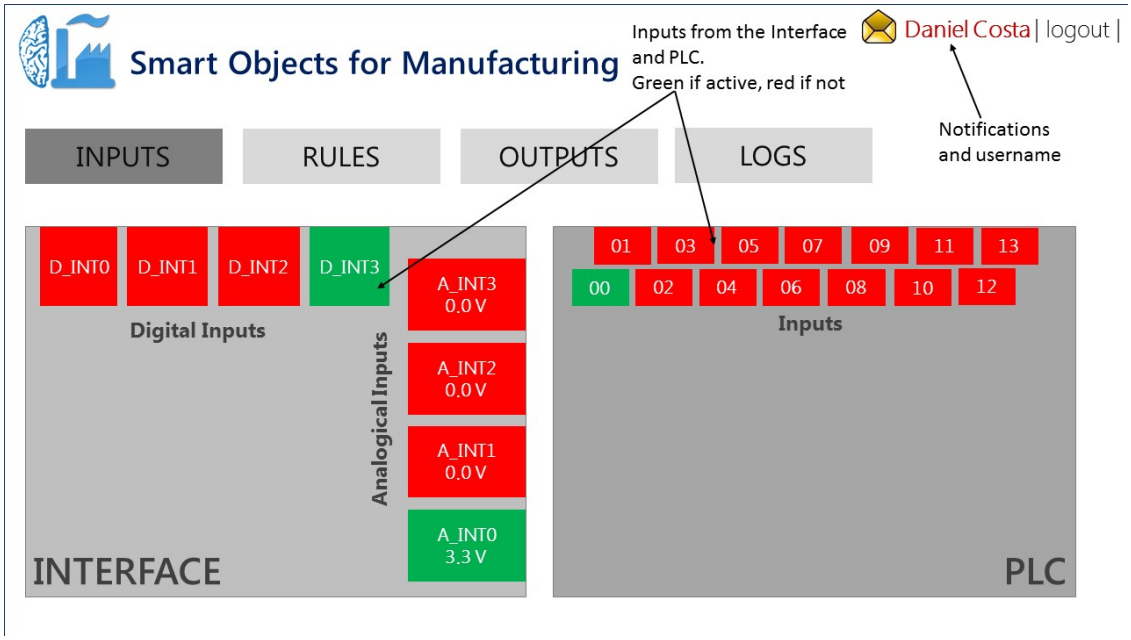


Figure D.7: Inputs Interface Screen

References

[1] Rosalie Zobel and Erastos Filos. Global collaborative environments for manufacturing innovation. *Proceedings of the IMS Vision Forum 2006, 12-13 April 2006*, pages 124–137, 2006.

[2] Prof. Umit Bititci, Dr Catherine Maguire and Dr Ian Gregory. Adaptive Capability A must for manufacturing SMEs of the Future, 2012. Available at: http://www.futuresme.eu/docs/research-reports/2011/01/26/adaptive-capability—a-must-for-smes.pdf?sfvrsn=3.

[3] The Economist. Foresight 2020 Economic, industry and corporate trends, 2006. A report from the Economist Intelligence Unit, sponsored by Cisco Systems.

[4] ADVENTURE. ADaptive Virtual ENTerprise manufacTURing Environment - Information Sheet, 2013. URL: http://www.fp7-adventure.eu/wp-content/uploads/2013/09/flyer2.pdf.

[5] TUDA, ASC, TIE, INESC, UVA, UVI, ISOFT, TANet, AZEV, and ABB. D2.1 project vision consensus document. *IEEE Internet Computing*, November 2011.

[6] Dave Evans. The Internet of Things - How the Next Evolution of the Internet Is Changing Everything. April 2011.

[7] Kevin Ashton. That 'Internet of Things' Thing, July 2009. RFID Journa.

[8] Jayavardhana Gubbia, Rajkumar Buyyab, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29:1645–1660, 2013.

[9] Gartner Inc. Gartner's hype cycle special report for 2011, 2012. URL: http://www.gartner.com/technology/research/hype-cycles/.

[10] Richcloud. Internet of Things (IOT), 2014. URL: http://www.shrct.com/en/index.php/solution/whitepaper/loginet.

[11] Michael Chu, Markus Löffler, and Roger Roberts. The Internet of Things. *McKinsey Quarterly*, March 2010. URL: http://www.mckinsey.com/insights/high_tech_telecoms_internet/the_internet_of_things.

[12] Libelium Comunicaciones Distribuidas. 50 Sensor Applications for a Smarter World, 2013. URL: http://www.libelium.com/top_50_iot_sensor_applications_ranking/.

[13] Ansgar Schlautmann, Didier Levy, Stuart Keeping, and Gregory Pankert. Smart market-makers for the “Internet of Things”. *Prism*, February 2011.

[14] Tomás Sánchez López, Damith C. Ranasinghe, Mark Harrison, and Duncan McFarlane. Using Smart Objects to build the Internet of Things. *IEEE Internet Computing*, Septmeber 2012.

[15] Ray’s DIY Electronics Hobby Projects. OpenSprinkler Beagle (OSBo), 2013. URL: http://rayshobby.net/?page_id=7664f.

[16] UnixMedia. OSSO – BeagleBone I/O Expansion Board, 2014. URL: http://www.unixmedia.net/schede-di-espansione-prototipazione/osso-beaglebone-io-expansion-board.

[17] Olimex. MOD I/O - INPUT OUTPUT EXPANDABLE BOARD WITH UEXT, 2013. URL: https://www.olimex.com/Products/Modules/IO/MOD-IO/open-source-hardware.

[18] Michael Leonard. How to choose the right platform: Raspberry pi or beaglebone black, 2013. URL: http://makezine.com/magazine/how-to-choose-the-right-platform-raspberry-pi-or-beaglebone-black/.

[19] Wapice. Wapice remote management, 2011. URL: http://www.wrm.fi/images/wapice_wrm_2011_net.pdf.

[20] Gerald Coley. *BeagleBone Black System Reference Manual*, revision a5.2 edition, April 2013.

[21] Wikipedia. Opto-isolator, 2014. URL: http://en.wikipedia.org/wiki/Opto-isolator.

[22] Texas Instruments Incorporated. HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON TRANSISTOR ARRAYS, 2014. URL: http://www.ti.com/lit/ds/symlink/uln2003a.pdf.

[23] Logic Supply, Inc. BeagleBone Black RS-232 Serial Micro-Cape, 2014. Model Number: CBB-TTL-232. URL: https://docs.google.com/document/d/1_mUWEFLprKzaQKn_pY9O2WO33k4wlFKaatJ2VKxDUS0.

[24] Simon Monk. SSH to BeagleBone Black over USB, 2014. URL: https://learn.adafruit.com/ssh-to-beaglebone-black-over-usb/overview.

[25] Omron. *Operational Manual*, 2012. Page 161-168. URL: http://www.miel.si/wp-content/VsebinaPDF/W516-E1-01+CP1L-EL(M)+UsersManual.pdf.

[26] Omron. FINS Communications, 2014. Section 5. URL: http://paginas.fe.up.pt/~pfs/recursos/plcs/omron/cs1/eth_manual/sec5.pdf.

[27] Omron. *Operational Manual*, 2012. Page 200. URL: http://www.miel.si/wp-content/VsebinaPDF/W516-E1-01+CP1L-EL(M)+UsersManual.pdf.

[28] TechTerms. Python, 2010. URL: http://www.techterms.com/definition/python.

[29] Tim Peters. *PEP 20 – The Zen of Python*. Python Software Foundation, 2004.

[30] Swaroop C H. A Byte of Pythons, 2005. Features of Python. URL: http://www.ibiblio.org/g2swap/byteofpython/read/features-of-python.html.

[31] Mike Driscoll. SQLite Python Tutorial, 2012. URL: http://www.blog.pythonlibrary.org/2012/07/18/python-a-simple-step-by-step-sqlite-tutorial/.

[32] UsingPickle (last edited 2014-06-04 13:32:28 by EricYe). Using Pickle, 2014. URL: https://wiki.python.org/moin/UsingPickle.

[33] Nixtutor). Send Mail Through Gmail with Python, 2009. URL: http://www.nixtutor.com/linux/send-mail-through-gmail-with-python/.